НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

«На правах рукопису»
УДК 004.4

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

Магістерська дисертація

на здобуття ступеня магістра
за освітньо-професійною програмою «Програмне забезпечення інформаційно-комунікаційних систем»
зі спеціальності 121 «Інженерія програмного забезпечення»
на тему: «Система пошуку закладів харчування в умовах нестабільної епідеміологічної ситуації»

Виконав:
студент VI курсу, групи ІТ-93мп
Войтенко Артем Віталійович _____

Керівник:
Доцент каф. АУТС, к. т. н., доц.
Полторак Вадим Петрович _____

Рецензент:
Завідувач каф. СПіСКС, д.т.н., доц.
Романкевич Віталій Олексійович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент_____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Програмне забезпечення інформаційно-комунікаційних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Войтенко Артем Віталійович

1. Тема дисертації «Система пошуку закладів харчування в умовах нестабільної епідеміологічної ситуації», науковий керівник дисертації Полторак Вадим Петрович, доцент каф. АУТС, кандидат техн. наук затверджені наказом по університету від «26» 10 2020 р. №3132-с

2. Термін подання студентом дисертації _____

3. Об’єкт дослідження: покращення взаємодії між відвідувачами і представниками закладів, забезпечення безпеки відвідування закладів харчування

4. Вихідні дані: Операційна система Windows 10, мова програмування JavaScript, бібліотека React, бібліотека Redux, мова розмітки веб-документів HTML, мова стилізації CSS, СУБД — MongoDB, Node.js

5. Перелік завдань, які потрібно розробити: виконати порівняльний аналіз систем аналогів, описати сценарії використання системи, визначити основні вимоги до системи, спроектувати архітектуру ситеми та реалізувати рішення.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: діаграма діяльності, карта шляхів користувача, діаграма розгортання, діаграма послідовності оформлення замовлення, діаграма послідовності взаємодії компонентів системи, діаграма варіантів використання застосунку для клієнтів закладів, діаграма варіантів

використання застосунку для адміністраторів закладів, діаграма діяльності, схема бази даних

7. Орієнтовний перелік публікацій: Войтенко А. В. Розробка сучасних веб-застосунків за допомогою технології PWA / Артем Віталійович Войтенко. // Winter InfoCom Advanced Solutions 2020. – 2020.

8. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисерта ції	Термін виконання етапів магістерської дисертації	Примітка
1	Видача завдання	07.09.2020	
2	Аналіз предметного середовища	10.09.2020 - 21.09.2020	
3	Формування вимог до системи	20.09.2020 – 30.09.2020	
3	Проектування архітектури системи	02.10.2020 – 16.10.2020	
4	Вибір набору технологій для реалізації системи	11.10.2020 - 15.10.2020	
5	Реалізації системи	15.10.2020 – 25.11.2020	
6	Оформлення документації	17.11.2020 – 02.12.2020	
7	Подання роботи до попереднього захисту	02.12.2020	

Студент Артем ВОЙТЕНКО

Науковий керівник Вадим ПОЛТОРАК

РЕФЕРАТ

Войтенко А. В. Система пошуку закладів харчування в умовах нестабільної епідеміологічної ситуації. КПІ ім. Ігоря Сікорського, Київ, 2020.

Робота містить 100 сторінок тексту, 16 рисунків, 33 таблиці, посилання на 16 літературних джерел та 8 додатків.

Актуальність теми обумовлена тим, що у зв'язку епідеміологічними умовами, що склалися у світі насьогодні, накладені обмеження на користування закладами громадського харчування мають негативний вплив на ресторанний бізнес і значно зменшують комфорт життя людей, що звикли харчуватися у закладах. Оскільки проблема постала відносно нещодавно, то рішень які могли б зменшити негативний вплив обмежень наразі не існує.

Метою роботи є мінімізація впливів епідеміологічної ситуації на сферу харчування як для представників бізнесу, так і звичайних людей, шляхом розробки програмного комплексу, що в сукупності буде являти платформу, на якій представник бізнесу, зможе розмістити свій заклад. Основна ідея полягає в тому, що потенційний відвідувач матиме змогу здійснити пошук закладу, в якому можна найбільш безпечно поїсти. Крім того, платформа дозволить здійснювати онлайн замовленя їжі, з метою реалізації послуги «заберу з собою», а також здійснення передзамовлення, для оптимізації часу, що витрачається на харчування.

У роботі проведено огляд існуючих рішень, сформовано список основних вимог та проведено аналіз найбільш актуальних технологій. Крім того описується сам процес розробки, використані патерни та архітектурні підходи.

В результаті розроблено програмний комплекс для взаємодії різних типів користувачів. Код серверного та клієнтських застосунків написаний на мові JavaScript. Клієнтські застосунки орієнтовані на роботу у браузерному середовищі, і являють собою односторінкові застосунки, що реалізовані з використанням бібліотеки Redux. Для роботи з даними було обрано СУБД MongoDB.

Ключові слова: система пошуку, односторінковий застосунок, Flux-архітектура, онлайн замовлення, прогресивний веб-застосунок .

ABSTRACT

Voitenko A. V. System for the search of food establishments under a bad epidemiological situation. Igor Sikorsky KPI, Kyiv, 2020.

The work contains 100 pages of text, 16 figures, 33 tables, links to 16 literary sources and 8 annexes.

The topic is actual as due to the epidemiological conditions in the world today, the restrictions on the use of catering establishments have a negative impact on the restaurant business and significantly reduce the comfort of life of people who used to eat in establishments. As the problem has emerged recently, there are no solutions that could reduce the negative impact of the restrictions.

The aim of the work is to minimize the impact of the epidemiological situation on the food sector for both business and ordinary people, by developing a few applications that together will be a platform on which business representatives will be able to place their establishment. The basic idea is to give to a potential visitor the instrument of search for the safest place to eat. In addition, the platform will allow users to order food online, for implementation the service «take away», and ability to make pre-order, for optimization of the time spent on lunch.

The paper reviews the existing solutions, forms a list of basic requirements and analyzes the most relevant technologies for developing own solution. The development process, patterns and architectural approaches are described.

As a result, a server and two client applications were developed, which together constitute a software for the interaction of different types of users. The code of the server and client applications is written in JavaScript. Client applications are browser-oriented, and implemented as Single Page Applications with usage of the Redux library. Document-oriented DBMS MongoDB was chosen to work with data.

Keywords: search system, single page application, Flux-architecture, online order, progressive web-application.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

SPA — Single Page Application;

SPI — Single Page Interface;

HTML — HyperText Markup Language;

CSS — Cascading Style Sheets;

ПЗ — Програмне забезпечення;

URI — Uniform Resource Identifier;

URL — Uniform Resource Locator;

HTTP — HyperText Transfer Protocol;

API — Application programming interface;

UX — User Experience;

UI — User Interface;

JSX — JavaScript XML;

DOM — Document Object Model;

СУБД — Система Управління Базами Даних;

JSON — JavaScript Object Notation;

PWA — Progressive Web Application;

ODM — Object Data Manager;

ВСТУП

На сьогоднішній день, перед світом постала проблема епідемії COVID-19, яка має негативний вплив майже на всі сфери бізнесу, суспільного життя та, зокрема, життя будь-якої людини. Один із напрямків, що постраждав більшою мірою— це сфера громадського харчування, адже відвідування закладів харчування, може бути причиною більш активного розповсюдження вірусу Covid-19 та, як наслідок, погіршення епідеміологічної ситуації вцілому. Для забезпечення максимальної безпеки відвідувачів та збереження бізнесу від банкрутства, підприємці прибігають до різних мір. Серед них: відкриття літніх терас, розділення місць у залах закладів на окремі «захищені кабінки» і тому подібне.

Крім цього, особливого дискомфорту зазнають люди зі щільним графіком у межах трафіку великого міста, що звикли харчуватися у закладах. У рамках обмежень, що застосовані для мінімізації ризику передачі вірусу, таких як обмеження кількості відвідувачів в залежності від площі закладу, їм доводиться витрачати багато часу для пошуку місця де можна безпечно та швидко поїсти. Оскільки, необхідно знайти заклад, відвідати його, щоб переконатися у тому, що, по-перше, забезпечені усі умови безпечного перебування, а, по-друге — що у закладі залишилися вільні місця. І якщо заклад не підходить, то потрібно шукати інший, що відповідає вимогам безпеки та має вільні місця. Отже, пошук закладу, стає проблемою для потенційного відвідувача і, як результат, може призвести до того, що він зовсім відмовиться від послуг закладів харчування. Це негативно впливає на дохідність ресторанного бізнесу, комфорт користувачів та економіку країни вцілому.

У зв’язку із накладенням обмежень на прийом гостей в приміщенні, багато закладів почали надавати послуги доставки своєї продукції. Проте, проблема полягає у тому, що попит на доставку сильно зріс, і часто доводиться досить довго чекати на замовлення, через високу завантаженість кур’єрських служб. І що робити у випадку невеликих міст, в яких немає агрегаторів доставки, формат кухні не передбачував доставку раніше або за розрахунками доставляти просто невигідно? Стає дедалі популярнішою послуга — «їжа з собою». Проте не всі заклади мають власні веб-сайти

з функціоналом, що дозволяє приймати і оброблювати замовлення. Через відсутність такої можливості, майже гарантовано, що потенційні клієнти оберуть інший заклад, замість того щоб робити замовлення по телефону. Тим більше прийом замовлень по телефону значно погіршує продуктивність, оскільки це займає набагато більше часу, і унеможливлює прийом декількох замовлень одночасно.

Навіть без врахування епідеміологічної ситуації і використання послуги «їжа з собою» — інтернет замовлення є відмінним рішення для підвищення лояльності постійних клієнтів, особливо якщо у меню є бізнес-ланчі. Передзамовлення популярне серед людей, які хочуть прийти на обід в улюблений заклад поруч з роботою. Вони не хочуть чекати, поки у них приймуть замовлення і приготують їжу. Все, що їм потрібно — це швидко і смачно поїсти, заощадивши час.

Під час оформлення замовлення гість вказує орієнтовний час свого приходу, щоб його страва була приготована безпосередньо перед приходом. Кухня працює ефективно, рівномірно розподіляючи навантаження в залежності від кількості замовлень.

На сьогоднішній день, не існує готових рішень, які б дозволяли власникам закладів налагодити процес інтернет замовлень у дуже короткий термін без розробки власного застосунку.

Отже, метою магістерської дисертації є розробка сервісу, який вирішить проблему пошуку безпечного закладу харчування і забезпечить можливість таким закладам інформувати потенційних клієнтів у реальному часі, стосовно відсотку заповнення закладу, вжитих заходів, що забезпечують безпеку відвідувачів, наявність терас та інше. А також, надасть можливості здійснення і обробки замовлень клієнтами і адміністраторами закладів відповідно.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

На сьогодні існує велика кількість сервісів, що дозволяють здійснювати пошук різних об’єктів за заданими критеріями, з можливістю використання геолокації користувача, для пошуку у найближчій доступності або певному радіусі. Усі вони надають інформацію про знайдені об’єкти, таку як: графік роботи, адресу, адресу веб-сайту, місцезнаходження, рейтинг, відгуки та інше. Не винятком є і пошук закладів харчування. Тільки на у «плей маркеті», представлено більше п’яти застосунків з подібним призначенням та функціоналом для андроід-девайсів, те ж саме можна спостерігати і для пристроїв на платформі IOS. Але жодне з існуючих рішень не орієнтоване на умови нестабільної епідеміологічної ситуації, і не надає необхідної інформації, стосовно наявності засобів забезпечення безпеки відвідувачів, що є важливим показником на сьогоднішній день. Також на ринку відсутні сервіси, що дававли б змогу здійснювати передзамовлення, або замовлення їжі «на винос». Усі існуючі рішення орієнтовані виключно на доставку.

Серед найбільш відомих рішень, що дозволяють здійснювати пошук закладів харчування можна відмітити Google maps, Tripadvisor та Restaurant Guru. Варто розглянути кожен сервіс окремо, для виявлення переваг та недоліків, на які варто звернути увагу при розробці власного рішення.

Google maps — найбільш розповсюджений безкоштовний застосунок, що представляє собою карту, із можливістю пошуку, і перегляду необхідної інформації про вибраний на ній об’єкт. Надає досить великий набір функцій, що включає пошук відповідно до геолокації, навігацію до об’єктів. У розрізі закладів харчування містить великий набір різної інформації, такої як часи відвідування, статистику відвідування, рейтинг, відгуки. Представлені версії для Android, IOS платформ та у вигляді браузерного застосунку. Скріншти інтерфейсу представлені на рисунку 1.1.

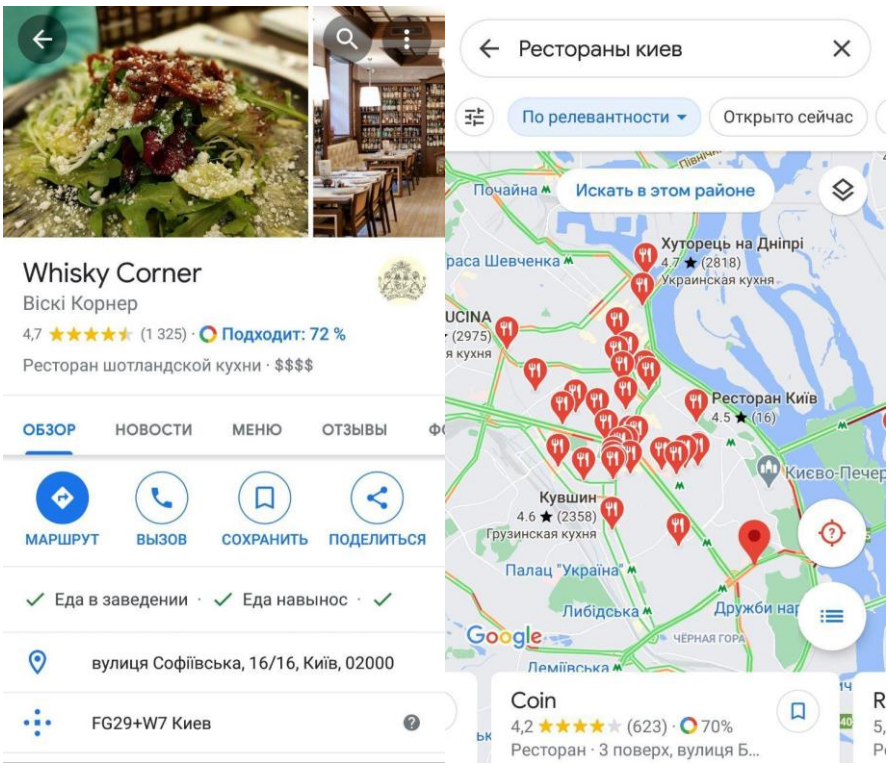

Рисунок 1.1 — Інтерфейс застосунку Google Maps

Переваги Google maps:

- кросплатформність;
- великий об’єм інформації;
- широкі можливості для пошуку;
- відносно актульна інформація;
- можливості навігації;
- відсутність реклами;
- підтримка більше 70 мов;
- охоплює більше 200 країн;

Недоліки:

- застосунок займає майже 700мб пам’яті;
- не зовсім зручний інтерфейс, оскільки застосунок має більш широке призначення, ніж просто пошук закладів харчування;
- сильне навантаження мобільних пристроїв;
- відсутність інформації про відсоток заповненості у реальному часі;

- відсутність інформації, що стосується роботи закладу в умовах несприятливої епідеміологічної ситуації;

- неможливість редагування інформації власниками закладу;

- недостатня структурованість.

Tripadvisor — безкоштовний застосунок, що призначений для пошуку усієї необхідної інформації, що стосується інфраструктури міста, необхідної для туристів, серед якої інформація про заклади харчування, готелі, розваги, оренду апартаментів. Надає широкі можливості для пошуку закладів харчування, із рекомендаціями, що засновані на реальних відгуках. На рисунку 1.2 наведені скріншоти застосунку.

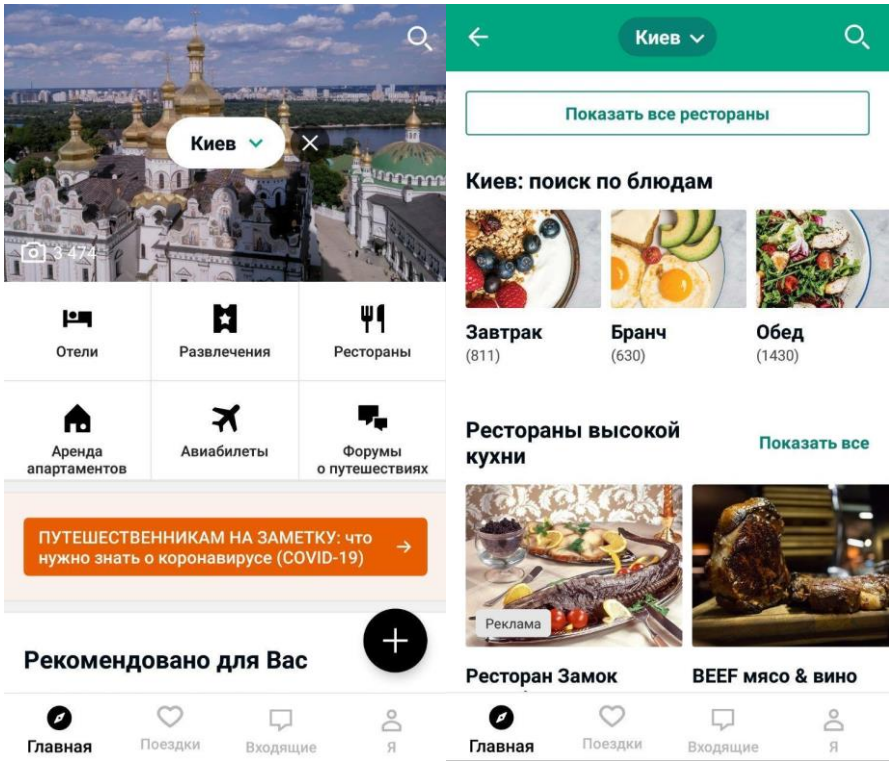

Рисунок 1.2 — Інтерфейс застосунку Tripadvisor

Переваги Tripadvisor:

- кросплатформність;

- приємний та зручний інтерфейс;

- широкі можливості для пошуку;

- реальний рейтинг та відгуки;

- структурованість інформації;

- зручне розбиття на категорії;

Недоліки:

- відсутність інформації, що стосується роботи закладу в умовах несприятливої епідеміологічної ситуації;

- неможливість редагування інформації власниками закладу;

- відсутність вбудованої навігації до закладу;

- наявність зайвого функціоналу, що погіршує зручність користування.

Restaurant Guru – безкоштовний мобільний застосунок для пошуку закладів харчування у будь-якому куточку світу. Скріншоти інтерфесу див. рисунок 1.3

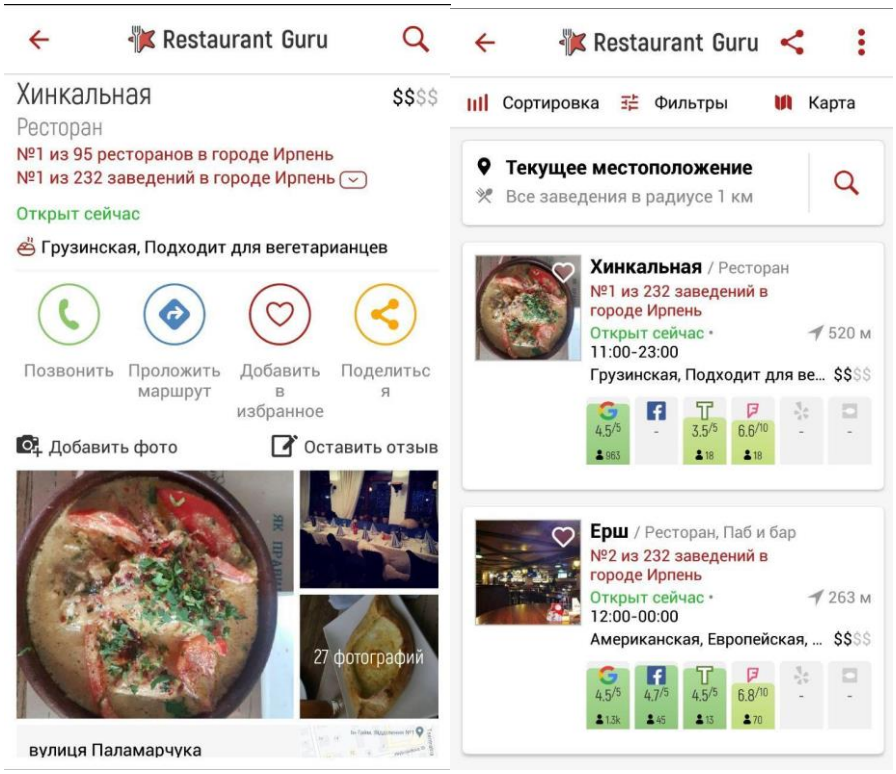

Рисунок 1.3 — Інтерфейс застосунку Restaurant Guru

Переваги Restaurant Guru:

- наявність офлайн режиму;

- пошук по стравах;

- наявність фільтрів, що допомагають зробити найбільш точний пошук відповідно до побажань користувача;

- наявність гастро-путівників,

- рейтингова оцінка із декількох найпопулярніших ресурсів.

Недоліки:

- відсутність інформації, що стосується роботи закладу в умовах несприятливої епідеміологічної ситуації;

- неможливість редагування інформації власниками закладу;

- відсутність можливості перегляду меню;

- незручний, перевантажений інтерфейс;

- застарілий дизайн та неякісна верстка;

- не завжди коректно спрацьовує фільтрація.

- Проаналізувавши існуючі рішення, їх переваги та недоліки, можна висунути такі критерії, на які треба зважити у при розробці власного продукту:

- застосунок не має бути перевантажений зайвим функціоналом;

- інтерфейс має бути простим та інтуїтивно зрозумілим;

- варто приділити увагу красивому візуально оформленню;

- має бути чітка структуризація закладів;

- реалізація системи рейтингу та коментарів;

- відсутність нав’язливої реклами;

- наявність зручної системи пошуку та фільтрації.

1.1 Висновки до розділу 1

Отже, у розділі було проведено аналіз предметної області та існуючих рішень. На основі цього аналізу можна сформувати подальший напрямок розробки та запозичити деякі, корисні ідеї, використані в існуючих продуктах. Також, у розділі описано деякі вимоги до розроблюваного програмного продукту, і виділено аспекти, на які варто звернути увагу при плануванні та розробці сервісу.

2 ПОСТАНОВКА ЗАДАЧІ ТА ФОРМУВАННЯ ФУНКЦІОНАЛЬНИХ І НЕФУНКЦІОНАЛЬНИХ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Технічне завдання

Ціль роботи передбачає розробку декількох застосунків, котрі будуть об’єднані в програмний комплекс, що дозволить здійснювати пошук закладів харчування і надавати максимально актуальну інформацію про них, заповнену безпосередньо представниками конкретного закладу, а також створювати та оброблювати замовлення. Також система передбачає можливість здійснення моніторингу відвідуваності, перегляду статистики, аналізу продуктивності роботи закладу, що може допомогти знайти підхід для покращення показників якості сервісу, та його прибутковості. Більш того, система дозволить створити велику інформаційну базу, інформація в якій буде регулярно оновлюватися.

Обов’язковим етапом у проектуванні систем є формування функціональних та нефункціональних вимог, адже без цього неможливо розробити правильну архітектуру майбутньої системи, та врахувати усі необхідні модулі. Також наявність вимог дає значний приріст у швидкості імплементації і дозволяє більш точно оцінити терміни розробки, оскільки основуючись на описаних вимогах, можна сформувати поетапний графік від початку написання коду, до реалізації MVP і подальшого розвитку продукту.

Так як планується реалізація декількох інтерфейсів, для різних типів користувачів (інтерфейс для потенційних клієнтів закладів харчування та для адміністрування з боку представників закладів), то варто сформувати окремі вимоги для обох клієнтських застосунків, а також окремо для серверної частини системи. З метою спрощення побудови архітектури в подальшому, було розроблено діаграми сценаріїв використання, що відповідають описаним функціональним вимогам, див. додаток А, та додаток Б. Оскільки основними функціями застосунку є пошук закладів і оформлення замовлення, було створено діаграму діяльності, що проходить по цьому функціоналу, див. додаток В.

2.2 Формування функціональних вимог

Функціональні вимоги — це вимоги до поведінки системи, вони визначають функціональність програмного забезпечення, тобто описують можливості, які надає система [1].

Функціональні вимоги визначають як необхідно реалізувати проект, включають в себе бізнес-вимоги і призначені для користувача вимоги. Їх можна згрупувати за трьома застосунками (застосунок для клієнтів закладу, застосунок для адміністрування з боку закладів, серверний застосунок).

2.2.1 Функціональні вимоги до застосунку, що призначений для клієнтів закладів харчування

Вимоги:

- наявність системи реєстрації/авторизації;
- реєстрація не обов'язкова;
- можливість швидкої авторизації за допомогою google аккаунта;
- система має пропонувати зберегти іконку на головному екрані пристрою, для швидкого доступу;
- користувач повинен мати можливість здійснювати пошук закладів відносно свого місця знаходження;
- користувач повинен мати можливість змінювати радіус пошуку закладів;
- у системі мають бути представлені фільтри для більш точного пошуку закладів, такі як відсоток заповненості, наявність тераси, обмеження за відстанню, фільтр за ціновою політикою, типом закладу;
- система має враховувати обмеження на користування закладами громадського харчування, накладені зонуванням у залежності від епідеміологічної ситуації;
- система має дозволяти здійснювати пошук по ключовим словам;

- система має представляти результати пошуку у вигляді списку, або відображати їх на карті;

- система пошуку має надавати можливості сортування за відсотком заповненості, рейтингом та іншими критеріями;

- має бути використана кольорова індикація відсотку заповненості, для більш швидкого сприйняття інформації;

- система має надавати можливість відображати інтегровану рекламу;

- система має надавати можливість додавати заклади до «обраних»;

- можливість перегляду меню, у зручному та чітко структурованому вигляді;

- користувач повинен мати можливість переглядати акційні пропозиції;

- при відображенні результатів пошуку, заклади, що мають акційні пропозиції повинні бути виділені відповідним чином;

- система має надавати можливість створення замовлення;

- можливість переглядати історію замовлень;

- можливість робити передзамовлення на певний час;

- можливість зробити замовлення їжі на винос;

- можливість оцінки якості виконання замовлення, із доданням коментаря.

2.2.2 Функціональні вимоги до застосунку, що призначений для адміністраторів закладу

Вимоги:

- застосунок має складатись з частини для заповнення інформації та частини моніторингу статистики відвідування;

- користувач повинен мати можливість прив’язати зареєстрований заклад до відповідного ідентифікатору закладу в Google Maps API;

- при перегляді статистики система повинна давати засоби для відображення статистики за певні проміжки часу;

- статистика має бути представлена у табличному вигляді та вигляді діаграм;

- система має вираховувати середню заповненість закладу в певні проміжки часу;

- у системі має бути присутній зручний конструктор меню, з можливістю додавання позицій, фотографій та цін;

- система повинна дозволяти додавати акційні та рекламні пропозиції і надавати можливості управління ними;

- система має надавати можливості для управління замовленнями.

2.2.3 Функціональні вимоги до серверного застосунку

Вимоги:

- надати REST API своїм клієнтам (застосунку для відвідувачів, та застосунку для представників закладу);

- забезпечити довгосторокове збереження даних;

- надати можливість додавання, видалення, редагування даних та їх передачі.

2.3 Формування нефункціональних вимог

2.3.1 Нефункціональні вимоги до застосунку, що призначений для клієнтів закладів харчування

Вимоги:

- має працювати на всіх можливих девайсах (мобільних телефонах, планшетах, комп’ютерах);

- можливість збереження іконки застосунку на екрані мобільного пристрою, завдяки використанню “web app manifest”;

- застосування принципів Progressive Web Application;

- наявність пуш нотифікацій;

- мінімальний час відгуку шляхом збереження нединамічної частини застосунку на стороні клієнта (App shell);

- застосунок повинен споживати не більше 4 Кбайт пам'яті на кожен неактивний сеанс з користувачем;

- підтримка клієнт-серверної архітектури. Застосунок повинен бути клієнтом програмного інтерфейсу, що наданий серверною частиною, він не повинен безпосередньо працювати з мережевою базою даних. Дана вимога забезпечить необхідний рівень ізоляції та безпеки даних, що зберігаються, а також, незалежність клієнтського застосунку від обраного сховища даних;

- дані на сервер повинні передаватися по захищеному каналу зв'язку щоб максимально знизити можливість перехоплення конфіденційних, особистих даних користувача;

- система має бути стабільною і стійкою до зовнішніх впливів, не повинно відбуватися аварійних або вимушених закриттів, зависань або інших аномалій в її роботі на будь-яких підтримуваних пристроях;

- система має бути легко масштабованою;

- система повинна мати можливість легкого відключення функціоналу, що безпосередньо пов’язаний із нестабільною епідеміологічною ситуацією у разі її повної стабілізації;

- застосунок повинен вимагати лише абсолютний мінімум прав доступу, які необхідні йому для підтримки основних функціональних можливостей;

- застосунок не повинен запитувати прав доступу до найбільш важливих даних;

- у застосунку не повинно залишатися жодних запущених служб при переході у фоновий режим, якщо це не потрібно для його основного функціоналу. Наприклад, застосунок не повинен залишати увімкненим модуль GPS;

- повинно забезпечуватися коректне збереження і відновлення стану користувача у застосунку;

- застосунок має зберігати стан користувача при переході у фоновий режим, запобігаючи випадковій втраті даних при навігації за допомогою кнопки «Назад» або при інших змінах стану. При поверненні з фонового в активний режим програма

має відновити збережений стан і всі пов'язані зі станом операції, які очікували виконання;

- коли застосунок викликається з перемикача останніх застосунків, має відновлюватися стан користувача на момент останньої роботи з цим застосунком;

- коли застосунок відновлюється після пробудження пристрою після блокування, застосунок повертає користувача в стан, відповідний до останнього моменту роботи з ним;

- коли застосунок повторно запускається з головного екрана або екрана застосунків, його стан повинен бути відновлено максимально близько до попереднього стану;

- при натисканні кнопки "Назад" застосунок дозволяє зберегти свій поточний стан або стан користувача, який в іншому випадку буде втрачено при переході назад;

- застосунок відображає графіку, тексти, зображення і різні елементи інтерфейсу без помітних спотворень, змазувань або ефектів пікселізації;

- для визначення координат повинен бути використаний відповідний програмний інтерфейс, що надається бібліотекою Google Play Services, що забезпечує максимально ефективне визначення поточного місцезнаходження, а саме низьку витрату батареї при досить точному (до 50 метрів в радіусі) визначенні місця розташування.

2.3.2 Нефункціональні вимоги до серверного застосунку

Вимоги:

- формат даних для обміну інформацією між сервером і клієнтськими застосунками повинен бути JSON;

- методи програмного інтерфейсу, що дозволяють записувати дані на сервер, які надаються серверної частиною, повинні бути доступні тільки для розроблюваних застосунків. Вони повинні бути захищені за допомогою аутентифікації на базі клієнтських SSL сертифікатів, тим самим, тільки довірені клієнти програмного інтерфейсу матимуть доступ на запис даних;

- для методів програмного інтерфейсу, які можуть повернути значний обсяг даних (більше кількох кілобайт), або які є ресурсоємними з точки зору отримання результату, забезпечити механізм кешування на рівні протоколу HTTP використовуючи директиву ETag: якщо клієнт запитує дані в API і вони не змінювалися з моменту попереднього запиту, то сервер повинен відповісти клієнту, що дані для поточного запиту не змінювалися клієнт може використовувати результати попереднього звернення до сервера. Це дозволить зменшити навантаження на серверний застосунок, а також зменшити обсяг переданої інформації;

- система повинна обслуговувати кожну сторінку в категорії «Важливі» не довше ніж 300 мсек (не включаючи затримки в мережі), за умови одночасного обслуговування не більше 5000 користувачів;

- система має підтримувати щорічний приріст користувачів на 15%;

- підтримувати щорічне зростання кількості транзакцій на 15% від попереднього значення;

- навантаження на CPU і наявний обсяг жорсткого диска на сервері баз даних не повинні перевищувати 70%, а час обробки запитів з категорії «Загальні» не повинно перевищувати 75 мсек, за умови одночасного обслуговування не більше 10 серверів;

- система має підтримувати можливість здійснення 500 запитів в секунду.

Спираючись на поставлені вимоги можна стверджувати, що для реалізації обох інтерфейсів найбільш оптимальним рішенням буде створення односторінкового браузерного застосунку (SPA) [2]. Аргументами такого вибору слугують наступні фактори:

- на сьогоднішній день більшість користувачів не любить встановлювати нативні застосунки з Play Market або App Store на свої пристрої, адже по-перше вони займають пам’ять, по-друге сам факт встановлення може відштовхувати потенційних клієнтів, оскільки це зайва низка дій, що віднімає час користувача;

- реалізація браузерного застосунку є більш універсальним рішенням, оскільки будь-який пристрій має браузер, і незалежно від того чи користувач

використовує девайс на платформі Android, IOS або Windows — він матиме змогу скористатися застосунком;

- розробка під браузер зазвичай займає набагато менше часу.

З точки зору дизайну можна частково використати концепцію односторінкового інтерфейсу — Single Page Interface. Це такий інтерфейс, що вміщується на одній сторінці і за свою мету має наблизити користування веб-застосунком до нативної програми.

Суть SPA полягає у тому, що сторінка не оновлюється при взаємодії користувача з нею, і навіть якщо користувач перейшов на іншу сторінку застосунку, зміна компонентів відбувається таким чином, що йому здається, ніби він лишається на тій же сторінці. Це позитивно впливає на враження від користування застосунком. При взаємодії з застосунком такого типу часто спостерігається динамічний зв’язок з веб-сервером.

Але варто врахувати той факт, що, все ж таки, досвід від користування нативним застосунком, справляє більш позитивне враження на користувача, аніж використання веб-сайту. Тому для вирішення цієї задачі, у застосунку, що призначений для клієнтів закладів, варто звернути увагу на такий підхід реалізації веб-застосунків, як Progressive Web Application.

2.4 Висновки до розділу 2

У даному розділі було сформовано функціональні та нефункціональні вимоги до усіх підсистем програмного комплексу, а також розглянуто основні концепції та правила побудови інтерфейсу для зручності користувачів. Основуючись на цих вимогах буде базуватися уся подальша розробка програмного комплексу.

3 ВИБІР ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ

3.1 Вибір і обгрунтування технологій для реалізації серверної частини

Фундаментом і основоположним кроком в розробці є визначення мови програмування, використовуваних інструментів і прийомів розробки. Варто зробити аргументований вибір майбутньої використовуваної мови програмування і інструментів для розробки системи, адже в подальшому це значний вплив на успіх реалізації і розвиток програмного продукту.

Було прийнято рішення використовувати за основну мову програмування JavaScript. Мова JavaScript дуже популярна завдяки присутності в будь-якому веб-браузері. Вона ні в чому не поступається іншим мовам програмування, але при цьому відповідає більшості сучасних вимог до мови програмування, що використовуються для типових рішень. Завдяки широкому поширенню є чимало досвідчених програмістів на JavaScript, що вплинуло на формування великого ком'юніті навколо даної мови. А велике ком'юніті в свою чергу створює умови для збільшення кількості бібліотек, розвитку мови і її популізації. JavaScript давно вийшла за рамки існування виключно контексту вікна браузера. JavaScript — це динамічна мова зі слабко типізованими, динамічно розширюваними об'єктами, які неформально оголошуються по мірі необхідності. Функції в ній є повноцінними об'єктами і зазвичай використовуються у вигляді анонімних замикань. Це робить JavaScript потужнішою мовою, в порівнянні з деякими іншими, часто застосовуваними для розробки веб-застосунків. Теоретично наявність подібних можливостей повинна підвищувати продуктивність розробки. Один з основних недоліків JavaScript — глобальний об'єкт. Це означає, що всі змінні верхнього рівня «звалюються» в глобальний об'єкт, і при використанні одночасно декількох модулів це може призвести до неконтрольованого хаосу. Оскільки веб-застосунок зазвичай складається з безлічі об'єктів, що можливо, створилися різними службами, то може виникнути ситуація, конфлікту глобальних об'єктів між собою. Але із застосуванням на стороні бекенд (сервера), Node.js використовує систему організації модулів CommonJS, це означає, що локальні змінні модулів так і будуть локальними, хоча і

виглядають як глобальні. Використання єдиної мови програмування на сервері і на клієнті є кілька потенційних плюсів:

- одні і ті ж програмісти можуть працювати над обома сторонами застосунку;
- код простіше переносити з сервера на клієнт і в зворотному напрямку;
- загальний для клієнта і сервера формат даних (JSON);
- загальний програмний інструментарій (як приклад IDE);
- загальні для клієнта і сервера засоби тестування і контролю якості;
- на обох сторонах веб-застосунку можна використовувати загальні шаблони представлень;
- спільна мова спілкування між групами розробників, що працюють над клієнтською і серверною частиною.

Node.js — відносно нова платформа для розробників веб-застосунків, серверів застосунків, довільних мережевих серверів і клієнтів, і програмування взагалом. Платформа спроектована так, щоб забезпечити високу масштабованість мережевих застосунків за рахунок продуманого поєднання асинхронного вводу / виводу, використання JavaScript на стороні сервера, винахідливого використання анонімних функцій JavaScript і однопоточної подієво-орієнтованої архітектури. Прийнята в Node.js модель принципово відрізняється від поширених платформ для побудови серверних застосунків, в яких масштабованість досягається за рахунок багатопоточності. Завдяки подієво-орієнтованій архітектурі [3] знижується споживання пам'яті, підвищується пропускна здатність і спрощується модель програмування.

На даний момент платформа Node.js дуже стрімко розвивається і багато розробників вважають її привабливою альтернативою традиційному підходу до розробки веб-застосунків.

Для створення гнучкого налаштування і швидкої розробки серверної частини програмного комплексу, доцільно вибрати відповідний фреймворк. В якості такого рішення був обраний Express, так як має наступні відмінні риси:

- висока швидкість обробки запитів, що підвищить швидкість роботи серверної частини вцілому;

- популярність і документація;
- готовий набір функцій для розробки SPA.

3.2 Вибір СУБД

Наступне важливе питання — яку систему управління базами даних краще всього використовувати в розробці системи.

Бази даних NoSQL добре підходять для багатьох сучасних застосунків, наприклад мобільних, ігрових, інтернет-застосунків, коли потрібні гнучкі масштабовані бази даних з високою продуктивністю і широкими функціональними можливостями, здатні забезпечувати максимальну зручність використання.

Гнучкість. Як правило, бази даних NoSQL пропонують гнучкі схеми, що дозволяє здійснювати розробку швидше і забезпечує можливість поетапної реалізації. Завдяки використанню гнучких моделей даних БД NoSQL добре підходять для частково структурованих і неструктурованих даних.

Масштабованість. Бази даних NoSQL розраховані на масштабування з використанням розподілених кластерів апаратного забезпечення, а не шляхом додавання дорогих надійних серверів. Деякі постачальники хмарних послуг проводять ці операції в фоновому режимі, забезпечуючи повністю керований сервіс.

Висока продуктивність. Бази даних NoSQL оптимізовані для конкретних моделей даних і шаблонів доступу, що дозволяє досягти більш високої продуктивності в порівнянні з реляційними базами даних [4].

Широкі функціональні можливості. Бази даних NoSQL надають API і типи даних з широкою функціональністю, які спеціально розроблені для відповідних моделей даних.

Два основні гравці на полі документних баз даних з відкритим вихідним кодом — MongoDB і CouchDB. Унаслідок більшого терміну життя, популярності, великої і відкритої документації варто обрати MongoDB. Отже, MongoDB — документоорієнтована система управління базами даних (СУБД) з відкритим вихідним кодом, яка не потребує опису схеми таблиць. Написана на мові C++. СУБД

MongoDB проектувалася для зберігання гігантських обсягів даних. Під час налаштування сервера Mongo перевага віддається узгодженості, після операції запису всі наступні операції читання витягнуть одне і те ж значення (до наступного оновлення). Ця особливість робить MongoDB привабливою альтернативою для тих, хто має досвід роботи з РСУБД. Крім того, MongoDB підтримує атомарні операції читання-запису, в тому числі інкрементування значення і запити до вкладених документів. Завдяки використанню JavaScript в якості мови запитів MongoDB підтримує як прості запити, так і складні завдання з розподілом-редукцією. В якості основних проектних цілей були поставлені висока продуктивність і простота доступу до даних. База даних дозволяє не тільки зберігати, але й опитувати вкладені дані, пред'являючи довільні запити. Схема бази даних не нав'язується, тому один документ може містити поля або типи, відсутні у всіх інших документах колекції. Документ Mongo можна уподібнити рядку реляційної таблиці без схеми, в якій допускається довільна глибина вкладеності значень. Отже, Mongo — відмінний вибір для зростаючого класу веб-проектів, в яких необхідно працювати з великими масивами даних, але бюджет занадто малий для придбання дорогого устаткування, або використовувати один підхід роботи з даними JSON + JavaScript. Очевидна і наочна структура зберігання даних має позитивний вплив на процес роботи з ними. Дані вводяться і виводяться в одному і тому ж вигляді. Завдяки відсутності структурованої схеми, Mongo може рости і змінюватися.

Так як в MongoDB відсутня схема опису зберігання даних, то опис предметної області проводиться шляхом інструменту моделювання об'єктів — Object Data Manager (ODM), в коді програми. В якості такого інструменту був обраний Mongoose, так як:

- має підтримку транзакцій;
- може перезапитує з'єднання з СУБД в разі втрати зв'язку;
- перевіряє на коректність тип даних, що зберігається;
- автоматично створює ідентифікатори вкладених сутностей;
- працює під Node.js.

3.3 Вибір технологій для реалізації клієнтської частини

HTML — стандартна мова розмітки, що використовується для побудови веб-сторінок. Основна її задача — це розміщення у документів зображень, мультимедійних файлів, текстів або таблиць. Основними елементами мови є HTML теги, які несуть в собі певне семантичне навантаження.

Разом з HTML при розробці буде використовуватися розширення синтаксису JSX, що поєднує у собі використання HTML тегів та JavaScript. Даний синтаксис надається можливостями бібліотеки React і по суті є зручним шаблонізатором.

3.3.1 Огляд технологій для стилізації

CSS — набір параметрів форматування (формальна мова), який застосовується до елементів документа, щоб змінити зовнішній вигляд документу, що написаний за допомогою мови розмітки. Дані параметри використовуються для задання розмірів блоків, їх розташування, зовнішнього та внутрішнього позиціонування текстів, зображень, задання кольорів та інше.

На сьогоднішній день невід’ємною частиною при написанні стилів сторінок є використання CSS препроцесорів, які значно спрощують синтаксис для стилізації документів. Це так званий синтаксичний цукор для розробників, котрий компілюється у звичайний CSS.

Переваги використання препроцесорів у порівнянні з «чистим» CSS:

- зручні можливості маніпулювання кольорами;
- можливість створення міксинів, для автоматизації певних процесів;
- можливість імпорту файлів;
- можливість проведення математичних операцій для обрахунку певних значень;
- можливість створення змінних;
- вкладати селектори один в одного, що позитивно впливає на зручність написання та читабельність коду.

Найбільш відомими та популярними препроцессорними мовами є: SASS, LESS, Stylus, Haml. Для реалізації візуального оформлення даного проекту обрано препроцесор SASS [5], оскільки він має зручніший синтаксис, має підтримку «чистого» CSS, а також має більш широкий спектр можливостей порівняно з іншими.

Крім того, у сучасній Front-end розробці, для покращення можливостей підтримки, масштабування і перевикористання певних блоків використовують CSS-методології, що засновані на деяких загальноприйнятих для конкретної методології правилах. Найпопулярнішими з них являються: BEM, SMACSS, ECSS. Однак найбільш розповсюдженим та ефективним рішенням являється БЕМ методологія, що буде застосована при реалізації проекту.

БЕМ — розшифровується як «Блок, Елемент, Модифікатор». Основний принцип даної методології полягає у розділенні інтерфейсу на незалежні блоки. Це дає можливість запобігати дублюванню коду шляхом перевикористання існуючих блоків. Важливі принципи, які передбачає BEM методологія [6]:

- відмова від використання ідентифікаторів для стилізації, оскільки це заваджує перевикористанню код;
- не застосовувати селектори тегів, оскільки HTML розмітка сторінок може змінитися, і при застосуванні тегу одразу зламаються стилі;
- не застосовувати універсальний селектор, оскільки даний селектор задає загальний стиль для проєкту, що впливає на усі вузли при верстці. Це накладає певні обмеження на повторне використання верстки в іншому проекті;
- не застосовувати вкладені селектори, бо такі селектори збільшують зв’язність коду і ускладнюють повторне використання;
- не застосовувати комбіновані селектори, так як комбіновані селектори збільшують специфічність і тому перевизначати блоки стає набагато складнішим;
- не суміщати клас і тег у селекторі, як і у випадку з комбінованими селекторами, збільшується специфічність;
- не застосовувати селектори атрибутів, оскільки вони є менш інформативними, аніж селектори по класам.

Отже, розробка з використанням методології БЕМ стає значно простішою та швидшою, підвищується читабельність та масштабованість кодової бази.

3.3.2 Вибір фронтенд фреймворку

Однією нефункціональних вимог, описаних у попередньому розділі є реалізація клієнтських застосунків як Single page application. SPA — це застосунок, який працює в браузері і не перезавантажує сторінку під час роботи. У SPA можуть використовуватися будь-які серверні технології. Оскільки значна частина веб-застосунку переміщується в браузер, вимоги до сервера можна істотно послабити.

SPA промальовується як персональний застосунок і перемальовує лише ті частини інтерфейсу, які змінилися, і лише тоді, коли це необхідно. Навпаки, традиційний сайт перемальовує всю сторінку у відповідь на різні дії користувача, що призводить до затримок і «миготіння», оскільки браузер повинен отримати сторінку від сервера і намалювати її на екрані. Якщо сторінка велика, сервер зайнятий або з'єднання з Інтернетом повільне, то затримка може скласти кілька секунд, а користувачеві залишається лише гадати, коли з нею знову можна буде працювати. Це великий недолік, в порівнянні зі швидкою промальовкою і миттєвим зворотним зв'язком SPA. SPA мінімізує час реакції за рахунок того, що переносить робочі (тимчасові) дані і частину обробки з сервера у браузер. У розпорядженні SPA є дані і бізнес-логіка, необхідні для прийняття більшості рішень локально, а значить, швидко. Лише аутентифікація користувача, валідація та постійне зберігання даних повинні залишитися на сервері. У разі традиційного сайту велика частина логіки застосунку знаходиться на сервері, тому, щоб отримати відповідь на свої дії, користувач повинен дочекатися завершення циклу запит-відповідь-рендрінг. Це може зайняти кілька секунд, тоді як реакція SPA майже миттєва. Якщо SPA все-таки має чекати відповіді сервера, то воно може динамічно оновлювати індикатор ходу виконання або зайнятості, щоб користувач не лякався затримки. При роботі ж з традиційним сайтом користувач змушений гадати, коли завантаження сторінки закінчиться і з нею можна буде взаємодіяти. На відміну від більшості нативних

застосунків, користувач може звернутися до SPA, маючи лише з'єднання з Інтернетом і пристойний браузер. В даний час все це є на смартфонах, планшетах, телевізорах, ноутбуках і настільних ПК. SPA, як і сайт, оновлюється і поширюється миттєво. Користувачеві взагалі не потрібно нічого робити, щоб отримати вигоду від цієї можливості — варто завантажити браузер, і все працює. Нативний застосунок зазвичай необхідно завантажити, а потім встановити, а інтервал між випуском версій може становити кілька місяців або навіть років. SPA, як і сайт, працює на різних платформах. Зазвичай ця особливість вважається перевагою для розробника, але вона не менш важлива для численних користувачів, що працюють з декількома пристроями або ОС, скажімо, з Windows вдома, з «Mac'ом» на роботі і з телефоном Android. SPA може запропонувати найкраще з обох боків — миттєву реакцію нативного застосунку разом з легкою доступністю сайту. JavaScript SPA доступне більш ніж мільярду пристроїв, які підтримують сучасні браузери і не потребують підключення сторонніх модулів. Саме перераховані плюси односторінкового застосунку перед звичайним сайтом та нативним застосунком роблять його оптимальним вибором для реалізації даного проекту.

У сучасному світі фронтенд розробки існує велика кількість різних JavaScript фреймворків та бібліотек, що призначені для реалізації SPA та вирішення такої важливої і непростої задачі, як синхронізація користувацького інтерфейсу і внутрішнього стану застосунку. Реалізація подібних інтерфейсів, що потребують синхронізації, на чистому JavaScript без використання допоміжних інструментів потребує набагато більший об’єм коду та затрат часових ресурсів. Крім цих проблем, постає також питання ефективності, та швидкодії. Головною ж проблемою являється необхідність оновлення користувацього інтерфейсу при кожній зміні стану застосунку. Кожне оновлення, потребує докладання неабияких зусиль, що виражаються відповідним кодом. Такий код не тільки важко писати й підтримувати, він також має сумнівну надійність, і його досить легко «зламати».

Отже використання інструменту для оновлення станів є практично обов’язковою ознакою сучасної Front-end розробки.

Серед найпопулярніших JavaScript інструментів можна виділити бібліотеку React.js, а також фреймворки Svelte, Ember, Angular.js та Vue.js.

При аналізі усіх недоліків та переваг найбільш популярних фреймворків для побудови SPA було прийнято рішення використовувати React.js, оскільки його концептуальні засади та архітектура найбільш відповідають вимогам даного проекту.

React.js — бібліотека від Facebook, яка являється домінуючою серед засобів для побудови SPA [2]. За допомогою даної технології можна безболісно створювати інтерактивні інтерфейси. Можна спроектувати прості представлення для кожного стану майбутнього веб-застосунку, і React зможе ефективно та продуктивно оновити і перемалювати тільки ті компоненти, які були порушені зміною даних.

Декларативні представлення роблять код більш передбачуваним при виконанні і більш зручним при налагодженні. При розробці можна створювати інкапсульовані компоненти, які керують своїм власним станом, а потім з'єднувати їх для подальшого використання і створення складних складових, призначених для користувача інтерфейсів.

Так як логіка компонентів написана на JavaScript, а не на мові-шаблонизаторі, програміст може легко передавати досить великий набір даних зі складною структурою по всьому застосунку, зберігаючи стан поза DOM. Завдяки використанню даної бібліотеки з'являється можливість завантажити клієнту відразу всі можливі «Представлення». Тобто для кожної дії користувача в веб-клієнті знайдеться відповідне графічне веб-предствлення, а з сервера потрібно завантажити лише те, що змінилося в даних (наприклад, якісь числові дані в моніторингу, або будь-яка інша інформація в зручному вигляді, наприклад, JSON). Завдяки такому підходу, при початковому завантаженні сторінки на веб-клієнт завантажується порівняно великий файл (близько 200-300 КБ для великих веб-сервісів), однак при подальших запитах в поточній сесії користувачеві завантажується від 1 Б до 2-3 КБ (в залежності від розміру змінених даних), при цьому на екрані перемальовується тільки змінена частина, що економить час завантаження необхідного представлення та Інтернет-ресурси користувача. В результаті спостерігається висока швидкість

роботи веб-сервісу, поліпшується реагування інтерфейсу, прискорюється перехід між сторінками завдяки клієнтському роутингу, а шаблонізація інтерфейсу значно спрощує доопрацювання клієнтської частини веб-застосунку. React побудований на концепції компонентів. Він відрізняється від таких фреймворків, як Angular або Ember, які використовують двосторонню прив'язку даних для оновлення HTML сторінки. React простіший інструмент для вивчення, ніж Angular або Ember — він набагато менший за розміром і добре працює з jQuery та іншими фреймворками. Він, до того ж, надзвичайно швидкий, так як використовує віртуальний DOM і оновлює тільки змінені частини сторінки (Звернення до DOM досі є самою повільною частиною сучасних веб-застосунків, тому дана бібліотека і отримує переваги в продуктивності, оптимізуючи його). DOM React має здатність створювати 200 000 вузлів в секунду, що значно перевищує середній показник вузлів для більшості сайтів. DOM може відтворювати зміни завдяки використанню алгоритму Diffing, який здатний скоротити обчислення різниці від складності з n3 до n.

Переваги React:

- у нього є потужна спільнота;

- для нього розроблено безліч сторонніх бібліотек, які допомагають вирішувати різні завдання;

- існують корисні додаткові компоненти для цієї бібліотеки;

- є розширення для браузерів, які допомагають налагоджувати застосунки, створені за допомогою даної бібліотеки;

- безліч документації та онлайн-ресурсів. Завдяки підтримці Facebook існує безліч можливостей використовувати документацію та онлайн-ресурси для навчання та використання React;

- високий рівень відклику та гнучкості;

- віртуальна DOM структура [7], яка дозволяє впорядковувати документи форматів HTML, XHTML або XML в дерево, яке найкраще підходить веб-браузерам для аналізу різних елементів веб-застосунку;

- функціонал React сприяє покращенню роботи UI.

Недоліки:

- необхідні засоби для збірки;

- через те, що бібліотека дуже стрімко розвивається, документація розміщується трохи хаотично;

- при спілкуванні з сервером необхідне складне асинхронне програмування.

Також варто зауважити, що React.js має власне розширення синтаксису JavaScript, що нагадує XML, і дозволяє використовувати синтаксис HTML тегів для рендерингу компонентів — JSX. Код написаний з використанням JSX компілюється у виклики методів бібліотеки React. На практиці це дуже зручний інструмент, що надає бібліотеці велику перевагу серед інших.

3.3.3 Огляд технологій, що супроводжують розробку на бібліотеці React.js

У шаблоні MVC сама бібліотека React відповідає за «View» складову, тому для управління складоваю «Model» прийнято рішення обрати бібліотеку Redux. Redux — інструмент для управління станом даних та станом інтерфейсу в JavaScript застосунках. Він підходить для SPA в яких управління станом з часом може ставати складним та заплутаним.

На сьогоднішній день для полегшення відладки та контролю розробки для більшості популярних технологій існують спеціальні браузерні розширення. Для спрощення користуванням Redux, відладки та відстежування зміни станів застосунку існує розширення для браузеру Google Chrome Redux DevTools. Воно дає можливість достатньо розгорнуто оглянути усі дії, що пов’язані з обміном даними, зміною властивостей компонентів та їх станів.

Також існує браузерне розширення React Developer Tools, яке доступне у Google Chrome, що дозволяє легко перевіряти ієрархію компонентів в інструментах розробника. Відповідно можна дослідити кореневі елементи, які були відрендерені на сторінці, а також дочірні компоненти. Вибравши один з компонентів у дереві, можна перевірити його поточні властивості та стан на панелі з правого боку.

3.4 Налаштування робочого середовища та підбір інструментів розробки

Налаштування робочого середовища це важливий етап у розробці будь-якого застосунку. Сучасні застосунки, розроблені на JavaScript мають досить розгорнуту структуру ресурсів та модулів, яка постійно ускладнюється, тому необхідно звертатися до засобів, що здійснюють збірку застосунку. На сьогодні найпопулярнішим рішенням для цього являється Webpack [8]. Webpack є досить швидким і потужним інструментом для збірки веб-проектів. Головна його особливість — це підтримка модульної системи. За допомогою нього можна організувати будь-яку логіку, будь-які формати, швидко зібрати проект, використовувати безліч інших сторонніх інструментів спільно з ним. Webpack збірку можна викликати двома способами: через консоль і через NPM скрипти. Логіка і поведінка того, як Webpack буде збирати проект і які перетворення він буде здійснювати описується в його файлі конфігурації. Webpack має низку вбудованих плагінів, а також додаткових, які без складнощів можна встановити. Багато з них допомагають під час розробки, а деякі потрібні також в режимі публікації.

Важливим моментом у розробці є дотримання єдиних домовленостей у організації структури проекту, і для кожного стеку певних технологій є свій набір вимог. Так як ці вимоги у всіх проектах з однаковими технологіями схожі, для спрощення організації і більш швидкого старту безпосередньої розробки завдяки позбавленню від рутинної роботи використовується деякий шаблон-генератор. У програмуванні це має назву «скаффолдинг». І так як для розробки було обрано фреймворк React, можна скористатися інструментом для швидкого старту проекту «create-react-app».

У Node.js існує вбудована підтримка управління пакетами, для якої застосовується інструмент NPM. Ідея модулів NPM — це набір загальнодоступних компонентів для багаторазового використання, які легко встановити через онлайн репозиторій; для них підтримується управління версіями і залежностями.

3.5 Висновки до розділу 3

У розділі було розглянуто та проаналізовано найпопулярніші технології, які підходять для реалізації поставлених завдань, а також допоміжні інструменти, що спрощують виконання тих чи інших задач. Отже, зважаючи описані у розділі на переваги та особливості різних інструментів, обрано наступний стек технологій для реалізації програмного комплексу:

- бібліотека React для роботи з відображенням користувацького інтерфейсу
- бібліотека Redux для створення архітектури обробки внутрішніх дій у клієнтських застосунках
- бібліотека Axios, як бібліотека для створення запитів до серверної частини
- Для вирішення завдань серверного застосунку було обрано такі технології:
- Node.js — платформа, що дозволяє виконувати JavaScript на сервері і надає можливості для взаємодії з системними ресурсами сервера;
- Express — фреймворк, що дозволяє реалізувати веб-сервер, маршрутизацію запитів користувачів, декомпозувати логіку програми на окремі підсистеми.
- Mongoose — інструмент моделювання об'єктів.

4 ПРОЕКТУВАННЯ І РЕАЛІЗАЦІЯ

4.1 Загальна архітектура програмного комплексу

Програмний комплекс виконаний за принципом клієнт-серверної архітектури і складається з трьох основних компонентів, діаграма розгортання наведена у додатку Г: серверна частина (сервер, надає API), застосунок для відвідувачів (клієнт API) і застосунок для адміністраторів (клієнт API), а також із аппаратного компоненту, що представлений у вигляді трекеру кількості відвідувачів закладу. Цей трекер може бути підключений до WI-FI і автоматично відправляти дані на вказаний сервер. Взаємодія між компонентами комплексу відбувається по протоколу HTTPS. HTTPS — це звичайний HTTP протокол, який працює за шифрованими транспортними механізмами SSL (в даному випадку TLS). Даний протокол забезпечує захист від прослуховування з'єднання і перехоплення даних, що пересилаються. На рівні застосунку серверна частина надає API за технологією REST [9]. Схема компонент комплексу зображена на рисунку 4.1.

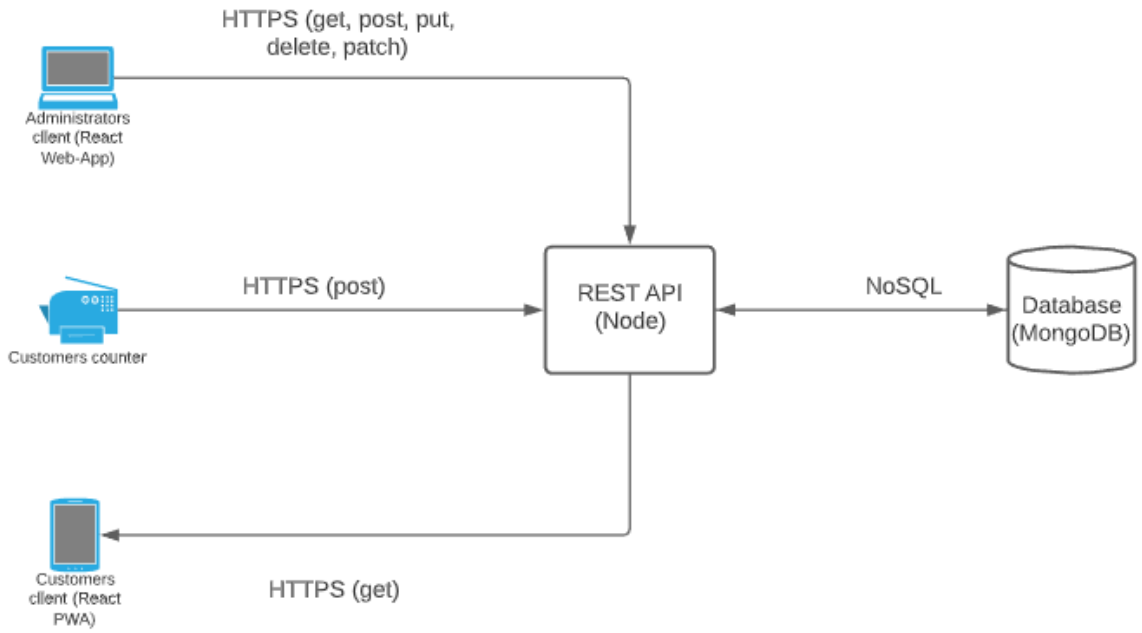

Рисунок 4.1 — Архітектура програмного комплексу

На схемі, стрілками вказано напрямки руху даних, і видно, що апаратна частина і застосунок для адміністраторів є джерелами даних, і відправляють їх на сервер для зберігання, а застосунок для відвідувачів є споживачем даних і відображає їх на інтерфейсі. Для отримання даних застосунок посилає запити на сервер до REST API. Діаграма послідовності, наведена у додатку Д, відображає взаємодію між застосунками програмного комплексу, на прикладі оформлення замовлення.

4.2 Вибір моделі обробки запитів

При виборі засобів реалізації, відзначимо такі особливості:

- очікується тенденція до зростання кількості запитів на одиницю часу;
- один запит не вимагає "великих" обчислювальних ресурсів сервера.

Через ці особливості прийнято рішення орієнтуватися на засоби по обробці запитів або виконання коду програми, які мають асинхронну модель роботи, засновану на подіях, або надають такі механізми. Це обумовлювалося тим, що асинхронна модель умовно дозволяє скоротити час відгуку застосунку, з тієї причини, що вона не блокує вхід-вихід і операційні завдання можуть виконуватися паралельно, підвищуючи загальну продуктивність застосунку на кожному рівні. Умовне порівняння швидкості виконання коду програми при асинхроній і синхронній моделі, показані на рисунках 4.2 і 4.3 відповідно. Асинхронна обробка запиту дозволяє уникнути затримки в роботі із застосунком. Користувачеві не потрібно чекати відповіді з сервера, і він може продовжувати виконувати різні дії на сторінці.

Потрібна інформація буде оброблена в фоновому режимі, і відповідь сервера оновить сторінку, коли вона надійде. Якщо станеться затримка відповіді, то користувачі цього навіть не помітять, так як вони будуть зайняті взаємодією з іншою частиною сторінки. Тому краще використовувати асинхронні запити.

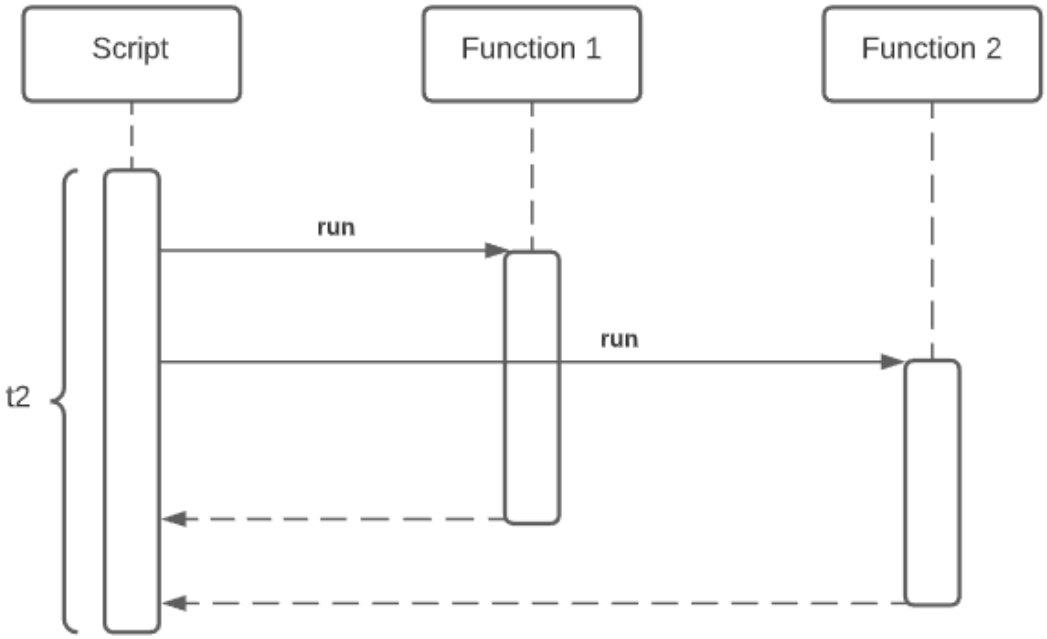

Рисунок 4.2 — Асинхронна модель обробки запитів

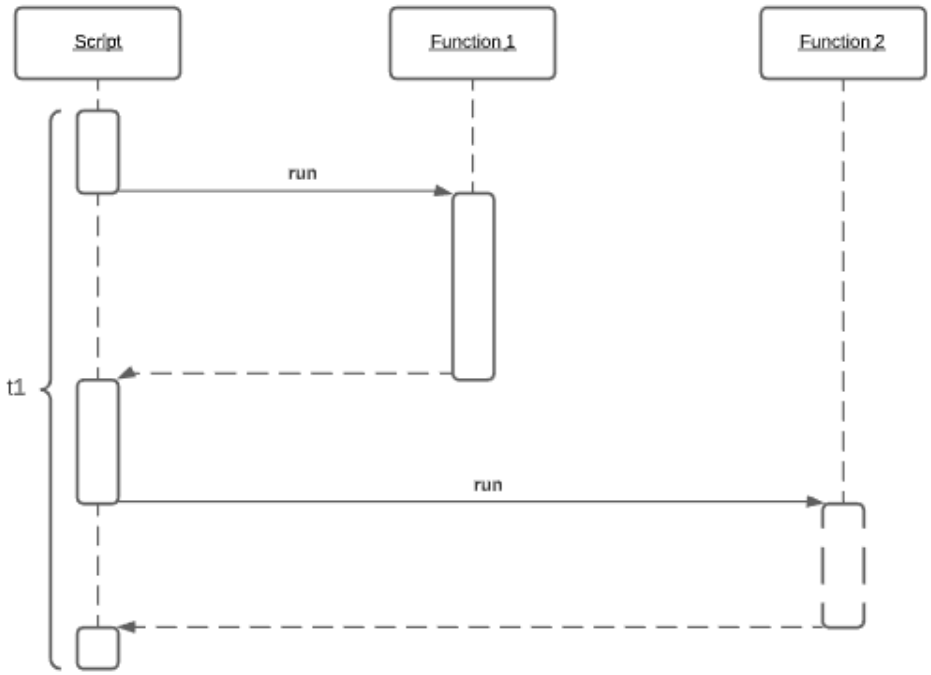

Рисунок 4.3 — Синхронна модель обробки запитів

З рисунків видно, що t1> t2. Це говорить про те, що час обробки запиту або виконання коду при асинхронної моделі менший.

4.3 Розподілення ролей

На звичайних веб-сайтах велика частка роботи по відображенню виконується сервером. Як наслідок, необхідно розробляти складні та важкі серверні застосунки, які повинні справлятися з досить високим навантаженням. В SPA дана частина роботи виконується на стороні клієнта, що дозволяє зняти із сервера значну частину навантаження і залишає для нього тільки сервісну частину. На рисунку 4.4 наведено розподілення ролей між сервером та клієнтом для традиційного веб-застосунку і SPA.

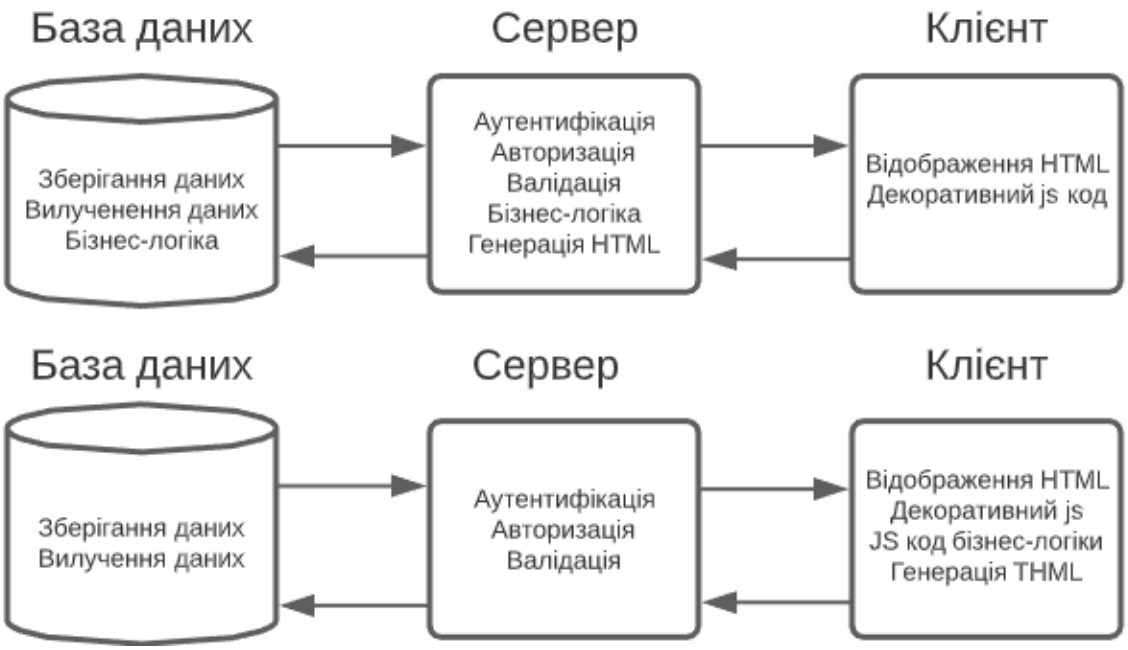

Рисунок 4.4 — Схема порівняння роботи SPA і традиційного веб-застосунку

4.4 Архітектура і розробка серверної частини

Архітектура серверної частини побудована на основі фреймворку Express. Він інкапсулює обробку високорівневих об'єктів застосунку, наприклад, таких як об'єкти запиту і відповіді, надаючи їх в розпорядження розробнику. Для структурування серверної частини системи були взяті принципи архітектурного типового рішення MVC.

4.4.1 Основні компоненти серверної частини

Основні компонентні представлення серверної частини системи:

- маршрутизатор (Route) відповідно до маршруту делегує подальшу обробку запиту керуючим об'єктам нижчого рівня: контролеру або API;

- контролер (Controller) відповідає за обробку запиту. Містить методи для реалізації функціональних вимог до системи. Використовує модель для отримання даних предметної області;

- модель (Model) відповідає за бізнес-логіку безпосередньо пов'язану з предметною областю. Є моделлю суті. Надає інтерфейс для роботи з сутностями. Інкапсулює обробку даних відповідної їй сутності. Mongoose — являє "обгортку" для моделей, надає функціональність для роботи з моделями;

- API представляє стандартизований інтерфейс для роботи з даними предметної області. Інкапсуляція бізнес-логіки за певним роутом. Повертає всі дані в форматі JSON. Запит до API визначається параметром «api» в рядку URL. Відповідно до REST, на прикладі роботи з документами, семантика запитів була побудована наступним чином, таблиця 4.1.

Таблиця 4.1 — Семантика запитів згідно REST

Ресурс\ Метод	GET	POST	PUT	DELETE
/document	Повернути список усіх документів.	Створити документ.	-	-
/document/:id	Повернути документ із вказаним ID	-	Оновити документ з вказаним ідентифікатором.	Видалити документ із вказаним ID

Аналогічно побудовані інтерфейси запитів і для інших сутностей. Для управління деякими сутностями була розроблена функція для перевірки доступу на виконання тих чи інших дій.

4.4.2 Основні класи серверного застосунку

Для реалізації необхідного функціоналу, були розроблені наступні класи і компоненти:

- AuthController — відповідає за логіку авторизації;
- EstablishmentsController — відповідає за усю логіку, що пов’язана з взаємодією із закладами;
- OrdersController — відповідає за логіку, що пов’язана із замовленнями;
- StatisticController — логіка, пов’язана зі статистикою;
- ReviewsController — логіка, пов’язана з відгуками;
- AdvertismentsController — логіка, пов’язана з рекламними пропозиціями;
- ErrorController — логіка обробки помилок;
- SearchController — логіка пошуку закладів;
- Email — клас, для формування і відправки електронних листів
- OrderModel — модель замовлення;
- EstablishmentModel — модель закладу;
- ReviewModel — модель відгуку;
- UserModel — модель користувача;
- StatisticModel — модель статистики відвідування;
- AdvertismentModel — модель рекламної пропозиції;
- HandlerFactory — клас, що містить хендлери для контроллерів.

Окоремо варто розглянути клас HandlerFactory. Він слугує для запобігання великої кількості дуплікацій коду, а також спрощення підтримки усього коду контроллерів. Клас містить такі методи як «getOne», «updateOne», «createOne», «deleteOne», «getAll». Ці методи використовуються як обгортка для методів різних контроллерів, і містять в собі усі повторювані частини, що відповідають за

зчитування, оновлення, створення, видалення даних відповідно. Це дозволяє легко вносити зміни, до усіх контроллерів одночасно, що значно спрощує підтримку коду, аджде в протилежному випадку при необхідності щось змінити, було б необхідно редагувати велику кількість методів та файлів. На рисунку 4.5 наведено лістинг коду, що відповідає за видалення відгуку, без застосування фабричного методу.

```
77  exports.deleteReview = catchAsync(async (req, res, next) => {
78    const review = await Review.findByIdAndDelete(req.params.id);
79
80    if (!review) {
81      return next(new AppError('No review found with that ID', 404));
82    }
83
84    res.status(204).json({
85      status: 'success',
86      data: null
87    });
88  });
```

Рисунок 4.5 — код видалення відгуку без застосування фабричного методу

З наведеного лістингу можна побачити що увесь код, крім конкроетної моделі, в даному випадку моделі «Review» буде повторюватися і в інших сутностей, для операцій видалення. На рисунку 4.6 наведено лістинг коду, що являє собою сам фабричний метод.

```
5   exports.deleteOne = Model =>
6     catchAsync(async (req, res, next) => {
7       const doc = await Model.findByIdAndDelete(req.params.id);
8
9       if (!doc) {
10        return next(new AppError('No document found with that ID', 404));
11      }
12
13      res.status(204).json({
14        status: 'success',
15        data: null
16      });
17    });
```

Рисунок 4.6 — код фабричного методу для видалення документів

З представленого лістингу зрозуміло, що фабричний метод, для видалення містить в собі увесь повторюючийся код, і тільки приймає конкретну модель в якості

аргументу. Це дає можливість дотримуватися принципу «DRY» (Don’t Repeat Yourself) і значно скорочувати код в контроллерах. Тепер можна використовувати даний фабричний метод у будь яких контроллерах для операцій видалення по id.

Після застосування фабричного методу, код видалення відгуку в контроллері виглядає наступним чином: «exports.deleteReview = factory.deleteOne(Review);». Тобто достатньо лише імпортувати клас «handlerFactory» в контроллер і викликати необхідний метод передавши відповідну модель.

4.4.3 Архітектура керована подіями

Більшість основних модулів Node.js, що використовуються на проекті, такі як «HTTP», «FileSystem», «Timers», побудовані навколо архітектури, керованої подіями. ЇЇ концепція полягає у тому що є об’єкти, які називаються «генераторами подій» (event emmiter), які викликають іменовані події як тільки в програмі відбувається щось важливе, наприклад, запит, що потрапляє на сервер, або таймер закінчується, або завершується читання файлу. Потім ці події можуть бути перехоплені «прослуховувачами подій» (event listeners), які реагують на події і виклакають відповідні «колбек» функції для певних подій. Найпростішим прикладом використання такої архітектури у застосунку є обробка запитів сервера в модулі «HTTP». У фалі «server.js» створюється змінна «server», що по суті є генератором подій, які надходять ззовні. Сервер автоматично видає подію під назвою "request" кожного разу, коли запит потрапляє на сервер. У файлі «app.js» реєструються прослуховувачі подій, що використовують певні роути, що імпортуються з директорії «Routes», в залежності від того який запит викликано. Самі роути якраз викликають «коллбек» функції, що являються методами відповідних контроллерів із директорії «Controllers». Під капотом сервер насправді є екземпляром класу Node.js — EventEmitter, тому він успадковує всю логіку генерації подій та прослуховування цих подій з цього класу. Логіка EventEmitter реалізовує відомий паттерн програмування «Observer». Отже, ідея полягає в тому, що «Observer», у цьому випадку є слухачем подій, який здійснює нагляд за суб’єктом, який згенерує подію, яку слухач чекає.

Використання цієї архітектури має великі переваги, які полягають в тому, що код є менш зв’язним у порівнянні з реалізацію через функції. Таким чином у коді застосунку відсутні, ситуації, коли функції із одного модулю викликають функції з іншого, оскільки це призвело б до безладу. Натомість ці модулі добре розділені та інкапсульовані в собі, кожен видає події, на які можуть реагувати інші функції, навіть якщо вони походять від інших модулів. Крім того, використання архітектури, керованої подіями, дозволяє простіше реагувати кілька разів на одну і ту ж подію.

4.5 Проектування бази даних

Для того щоб правильно спроектувати базу даних, було прийнято рішення почати з концептуального проектування. Для цього було виділено всі сутності і зв'язки в системі, відображено на рисунку 4.7.

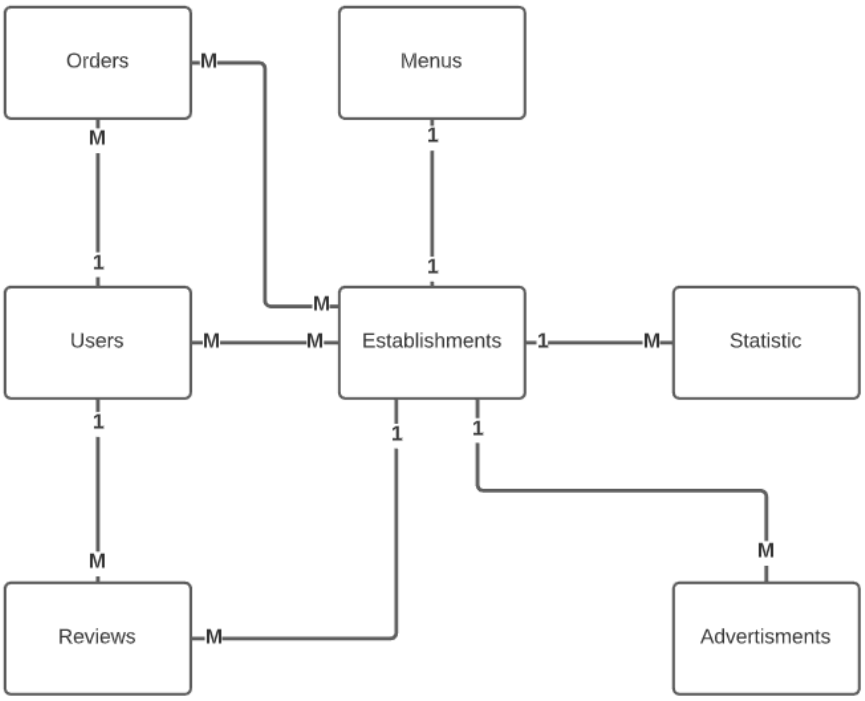

Рисунок 4.7 — Схема зв’язків сутностей бази даних

З рисунку 4.7 видно в системі є сім основних сутностей:

- users (дані користувачів);
- establishments (дані закладів);
- menus (дані меню);
- reviews (дані відгуків);
- statistic (дані про кількість відвідувачів закладу);
- advertisements (дані спеціальних пропозицій);
- orders (дані замовлень).

При проектуванні NoSQL бази даних важливо, крім визначення зв’язків між сутностями, правильно підібрати форму представлення для кожної сутності, що передбачає поділ сутностей на нормалізовані (referenced) та ненормалізовані (embedded). Тобто сутності можуть існувати як окремі колекції, або бути вкладеними в інші сутності. А також, для нормалізованих сутностей визначити тип зв’язування (child referencing, parent-referencing або two-way referencing), що і було виконано, результати наведені у таблиці 4.2.

При проектуванні були визначені деякі особливості проектування схеми БД в MongoDB, які полягають в тому, що:

- зберігання тільки ідентифікатора об'єкта зовнішньої сутності потребує виконання додаткового запиту для отримання всіх даних зовнішньої сутності;
- вкладення об'єкта сутності має сенс тоді, коли вкладений об'єкт буде зустрічатися разом з батьківським;
- вкладення об'єкта сутності відображає в реляційній схемі зв'язок один до багатьох;
- посилання забезпечують більшу гнучкість при частих змінах вкладеної сутності;
- масив посилань об'єктів сутності відображає в реляційній схемі зв'язок багато до багатьох.

Таблиця 4.2 — Визначення зв’язків між сутностями

Сутність 1	Сутність 2	Embedding / Referencing	Referencing type
Users	Establishments (parent)	Referencing	Child-referencing
Reviews	Users (parent)	Referencing	Parent-referencing
Establishments (parent)	Reviews	Referencing	Parent-referencing
Advertisements	Establishments (parent)	Referencing	Parent-referencing
Orders	Establishments (parent)	Referencing	Parent-referencing
Orders	Users	Referencing	Child-referencing
Statistics	Establishments (parent)	Referencing	Parent-referencing
Menus	Establishments (parent)	Embedding	-

Для зв’язку Users-Establishments обрано тип Child-referencing, оскільки один користувач, з роллю адміністратора може бути прив’язаний до декількох закладів (наприклад, заклади однієї мережі). Цей тип зв’язку підходить найбільше, так як, кількість закладів обмежена, і немає ризику переповнення. Передбачається, що модель користувача буде містити в собі масив ідентифікаторів закладів.

Вибір типу Parent-referencing для зав’язку Users-Reviews аргументується тим, що, по-перше, відгуки можуть використовуватися окремо від користувача, тому немає сенсу вкладати їх, а по-друге, припускаються операції зміни, окрім читання для

даної сутності. Отже для зв’язування цих сутностей модель відгуку повинна містити в собі ідентифікатор користувача, що його створив.

Тип зв’язку Parent-referencing для сутностей Review та Establishment пояснюється тим, що один заклад, може мати безліч відгуків. Цей факт суперечить використанню вкладення відгуків всередину закладу, а також збереженню усіх ідентифікаторів відгуків всередині екземпляру закладу. Тому модель сутності відгуку включатиме в себе посилання на заклад.

Оскільки заклад може мати лише декілька спеціальних пропозицій, і в той же час їх дані можуть бути використані окремо від закладу, то тип для зв`язування сутностей Advertisement і Establishment визначений як Parent-referencing. Модель рекламної пропозиції буде містити посилання на заклад.

Сутність Orders зв’язана з Establishments зв’язком Parent-referencing, так як кожен заклад може мати велику кількість замовлень, отже немає сенсу зберігати масив посилань на них всередині батьківської сутності.

Orders і Users зв’язані типом Child-referencing, оскільки користувач може мати обмежену кількість замовлень.

Для зв’язку Statistics-Establishments обрано тип Parent-referencing, оскільки кількість записів про число відвідувачів може бути для кожного закладу достатньо великою, тому є сенс включення у модель запису ідентифікатору закладу.

Тип зв’язку Parent-referencing для сутностей Statistics та Establishment пояснюється тим, що один заклад, може мати безліч записів про кількість відвідувачів. Цей факт суперечить використанню вкладення відгуків всередину закладу, а також збереженню усіх ідентифікаторів записів всередині екземпляру закладу. Тому модель сутності записів включатиме в себе посилання на заклад.

Сутність Menu вкладена у Establishments, оскільки може бути тільки одне меню для закладу і функціонал не передбачає частого редагування та окремого використання.

4.5.1 Опис колекцій бази даних

Як було вказано раніше, база даних має 7 сутностей, кожна з яких має відповідну модель. Для того щоб мати уявлення про відносини між сутностями, а також поля, що властиві для відповідної моделі кожної з них, була розроблена схема БД, наведена у Додатку Е на якій відображені псевдо-зв'язки.

Users. Дана колекція зберігає інформацію про зареєстрованих у системі користувачів. Структура наведена у таблиці 4.3.

Таблиця 4.3 — структура колекції Users.

Поле	Тип даних	Опис
_ID	ObjectId	унікальний ідентифікатор
login	String	Логін користувача
email	String	Електронна адреса користувача
phoneNumber	String	Телефон користувача
passwordHash	String	Пароль
refreshToken	String	Токен для зміни паролю
verificationToken	String	Верифікаційний токен
isAdmin	Boolean	Роль користувача
establishments	Array	Заклади, до адміністрування яких користувач має доступ

Orders. Дана колекція відповідає за зберігання даних стосовно здійснених замовлень. Структура колекції наведена у таблиці 4.4.

Таблиця 4.4 — структура колекції Orders

Поле	Тип даних	Опис
_ID	ObjectId	унікальний ідентифікатор
userId	ObjectId	Ідентифікатор користувача, що створив замовлення
establishmentId	ObjectId	Ідентифікатор закладу, у якому зроблено замовлення
orderList	Array	Замовлені одиниці
createdAt	date	Дата створення замовлення
updatedAt	date	Дата оновлення замовлення
review	Nested Object	Відгук про якість виконання замовлення
timeToComplete	integer	Розрахунковий час на виконання замовлення
preOrderTime	date	Час на який зроблено передзамовлення
status	String	Статус замовлення
takeAway	Boolean	Індикатор використання послуги «Їжа з собою»

Establishments. Колекція зберігає інформацію про заклади харчування. Структура наведена у таблиці 4.5.

Таблиця 4.5 — структура колекції Establishments

Поле	Тип даних	Опис
_ID	ObjectId	унікальний ідентифікатор
name	String	Назва закладу
googlePlaceIdentifier	String	ідентифікатор закладу у базі Google places
awards	Array	Нагороди та сертифікати
menu	Object	Меню
location	GeoJSON Object	Геопозиція закладу
contactInfo	Object	Контактна інформація
description	String	Короткий опис
priceLevel	Number	Цінова політика закладу
cuisine	Array	Кухня, представлена у закладі
pandemicParams	Object	інформація, що стасується роботи закладу під час пандемії
totalCapacity	Integer	Загальна місткість закладу

Reviews. Колекція зберігає відгуки про заклади. Структура наведена у таблиці 4.6

Таблиця 4.6 — Структура колекції Reviews

Поле	Тип даних	Опис
_ID	ObjectId	унікальний ідентифікатор
reviewText	String	Текст відгуку
rating	Integer	Оцінка
userId	ObjecId	Ідентифікатор користувача
establishmentId	ObjectId	Ідентифікатор закладу

Menu. Дана колекція вкладена в колекцію закладу, і містить в собі інформацію про доступні позиції в меню закладу. Структура документу наведена у таблиці 4.7.

Таблиця 4.7 — структура документу колекції Menu

Поле	Тип даних	Опис
_ID	ObjectId	унікальний ідентифікатор
itemName	String	Назва страви
ingredients	String	Інгрідієнти
category	String	Тип страви
price	Integer	Ціна за порцію
weight	Integer	Вага порції
description	String	Опис страви
establishmentId	ObjectId	Ідентифікатор закладу

Advertisment. Колекція для зберігання спецпропозицій закладів. Структура описана в таблиці 4.8.

Таблиця 4.8 — структура колекції Advertisment

Поле	Тип даних	Опис
_ID	ObjectId	Унікальний ідентифікатор
establishmentId	ObjectId	ідентифікатор закладу, якого стосується спецпропозиція
description	String	опис спецпропозиції
startDate	Date	дата початку дії спеціальної пропозиції
endDate	Date	дата закінчення дії спеціальної пропозиції
image	String	адреса зображення для спецпропозиції

Statistic. Колекція для зберігання статистики кількості відвідувачів. Структура описана в таблиці 4.9.

Таблиця 4.9 — Структура колекції Statistic

Поле	Тип даних	Опис
_ID	ObjectId	унікальний ідентифікатор
establishmentId	ObjectId	ідентифікатор закладу
date	Date	дата запису
visitorsCount	Integer	кількість відвідувачів закладу на момент запиту

4.6 Робота з геоданими

Функціональні вимоги передбачають такі функції як пошук закладів у певному радіусі у залежності від геопозиції користувача, а також вирахування відстані до закладу. Крім того, згідно вимог має бути реалізовано сортування за відстанню до закладу.

Кординати зберігаються у базі даних у вигляді «GeoJSON» об’єктів, які підтримуються MongoDB. У загальному вигляді такий об’єкт виглядає наступним чином:

```
<field>: {
        type: <GeoJSON type>,
        coordinates: <coordinates>
    }
```

де «type» — це тип GeoJSON об’єкту, і «coordinates» відповідно координати. В залежності від типу, може бути різний набір координат. Для зберігання місцязнаходження закладів харчування використано тип «Point», що по суті являє собою точку, із заданими значеннями довготи та широти. Це дозволяє використовувати так звані «геопросторові запити» (Geospatial Queries) представлені MongoDB. Для вирішення завдань розроблюваної системи були використані оператори $geoWithin та $centerSphere, а також $near.

4.7 Безпека даних

Паролі користувачів не зберігаються в системі у відкритому вигляді. У базі даних зберігаються результати обчислення криптографічної хеш-функції bcrypt від пароля. Bcrypt — адаптивна криптографічна функція формування ключа [10], що використовується для захищеного зберігання паролів. Для захисту від атак за допомогою райдужних таблиць [11] bcrypt використовує сіль, тобто довільний рядок даних, що додається до рядка пароля. У разі успішної аутентифікації клієнтського застосунку, сервер висилає застосунку згенерований JWT [12], який

використовується в подальшому для авторизації клієнтського застосунку. JWT (JSON Web Token) — це відкритий стандарт для забезпечення безпеки передачі пакетів між сторонами. Токен передається в зашифрованому вигляді і складається з трьох частин, між якими ставлять крапку:

- закодований заголовок, що містить інформацію про тип стандарту («JWT») і про алгоритм шифрування;

- тіло токена, яке містить метадані. У розроблюваних застосунках тіло токена містить тип клієнту (застосунок для відвідувачів закладів або застосунок для адміністраторів), ідентифікатор клієнтського застосуку і дату закінчення токена, після якої токен буде вважатися недійсним;

- цифровий відпис, який містить ключ для шифрування всіх частин токена.

Щоб браузер міг зберегти токен безпечним способом він передається у захищеному файлі cookie. В класі authController є метод, . createSendToken, який відповідає за створення токену і викликається у методі «signup» і «login», які відповідають за реєстрацію і авторизацію відповідно. Cookie — це невеликий шматок тексту, який сервер може надіслати клієнтам. Коли клієнт отримує файл cookie, він автоматично зберігає його, а потім автоматично відправляє його назад разом із усіма запитами до того ж сервера. Для відправки cookie достатньо прикріпити його до відповіді. Разом з токеном відправляється термін зберігання cookie, що становить 90 днів після чого він автоматично видалиться з браузеру.

За вихід користувача із системи відповідає метод «logout», який в свою чергу встановлює для cookie термін закінчення 10 секунд від поточного часу. Після чого токен видалиться із браузеру.

Для запобіганню тому, що одна і та ж IP адреса надсилає занадто багато запитів до API, застосовано стратегію обмеження швидкості, що допомогає запобігти таким атакам, як «Denial of service» або «Brute force attack» [13]. Цей обмежувач швидкості реалізований як глобальний мідлвар. Функція обмежувача швидкості підраховує кількість запитів, що надходять від одного IP, а потім, коли запитів занадто багато, блокує ці запити. Тому має сенс реалізувати це в як мідлвар у файлі «app.js». У функцію передається об’єкт, що містить в собі обмеження 300

запитів за годину. Якщо цю межу перетне певний IP, буде повернено повідомлення про помилку.

Для захисту від NoSQL ін’єкцій застосовано санітизацію даних, що дозволяє видалити небезпечні символи, що надходять від хакерів. Для цього також використовується мідлвар, в якому викликається функція, імпортована з пакету «express-mongo-sanitize». Даний мідлвар переглядає тіло запиту і сам рядок запиту, відфільтровуючи всі знаки і крапки доларів, оскільки вони використовуються в операторах MongoDB.

4.8 Архітектура клієнтських застосунків

Клієнтська частина являє собою архітектурне рішення "Товстий клієнт", що передбачає функціональний користувальницький інтерфейс, де і виконується частина бізнес-логіки. У клієнтську програму було вирішено винести ту бізнес-логіку, яка не вимагає звернення до БД. Клієнтський застосунок працює в браузері з об'єктною моделлю документа (DOM). DOM є програмним інтерфейсом для доступу до документів. щоб організувати роботу належним чином з DOM, API серверної частини і іншим, як правило, потрібні об'єкти JavaScript відповідальні за бізнес-логіку, логіку управління, логіку представлення [11]. У процесі розробки, були відзначені деякі особливості клієнтської частини:

- розробка логіки клієнтської частини може вестися незалежно від серверної частини;
- при інтерпретації HTML клієнт знаходить вказівку на те, які файли JavaScript застосунку необхідно запросити;
- JavaScript перевизначає деякі стандартні події DOM, наприклад, події переходу за посиланнями, відправки форм і т.п. Перехоплення JavaScript стандартних подій використовується для виконання тієї чи іншої логіки;
- HTML спочатку порожній, без даних. Дані запитуються у API на сервері. JavaScript-застосунок використовує можливість виконати запит на сервер за

допомогою об'єкта XmlHttpRequest. Після відповіді, клієнтський застосунок вирішує, що необхідно виконати на клієнті;

- дані клієнтського застосунку зберігаються в пам'яті.

4.8.1 Архітектурний підхід Flux

Описаний вище відхід до розробки клієнтської частини передбачає дуже тісну взаємодію з даними у більшості основних компонентів застосунку, а також постійну зміну станів. Для вирішення питання передачі даних, отриманих з бази до компонентів, для подальшого їх використання і обробки, а також питання передачі властивостей та станів між компонентами доцільно звернутися до певного архітектурного підходу.

Так як застосунок працює з динамічними даними, при реалізації клієнтських складових програмного комплексу за основу було взято архітектурний підхід Flux [12]. Flux включає в себе набір патернів програмування для побудови користувацького інтерфейсу, що поєднується з реактивним програмуванням.

Основною відмінною рисою Flux є одностороння спрямованість передачі даних між компонентами Flux-архітектури. Архітектура накладає обмеження на потік даних, зокрема, виключаючи можливість оновлення стану компонентів самими собою. Такий підхід робить потік даних передбачуваним і дозволяє легше простежити причини можливих помилок в програмному забезпеченні. У першу чергу Flux працює з інформаційною архітектурою, яка потім відбивається в архітектурі програмного забезпечення, тому рівень представлень слабо зв’язаний з іншими рівнями системи.

В даний час однією з найбільш популярних реалізацій Flux-архітектури є бібліотека Redux, що і використана при розробці застосунків. У Redux загальний стан застосунку представлено одним об'єктом JavaScript — state (стан) або state tree (дерево станів). Для работы с Redux в React необхідні залежності «redux» и «react-redux». Крім цього того, для работы с данными використовується структура даних Immutable.Map, тому також додана залежність Immutable.js Будь-які зміни даних, створених за допомогою цих структур даних, повертають новий об’єкт, який є

результатом змін. Крім того, що не потрібно турбуватися про випадкову мутацію стану застосунку, структури даних Immutable.js є високопродуктивними завдяки реалізації бібліотекою структурного обміну за допомогою хеш-карт та векторних спроб.

Основними складовими компонентами взаємодії Flux архітектури є Actions, Dispatcher, та Stores та Contorller Views.

Actions (Дії) — хелпери, що спрощують передачу даних Диспетчера.

Dispatcher (Диспетчер) — функція, що приймає Дії і розсилає навантаження зареєстрованим обробникам.

Stores (Сховища) — контейнери для збереження стану застосунку та бізнес-логіки в обробниках, зареєстрованих в диспетчері

Controller Views (Представлення) — React-компоненти, які збирають стан сховищ і передають його дочірнім компонентам через властивості (props). Даний процес представлений на рисунку 4.8.

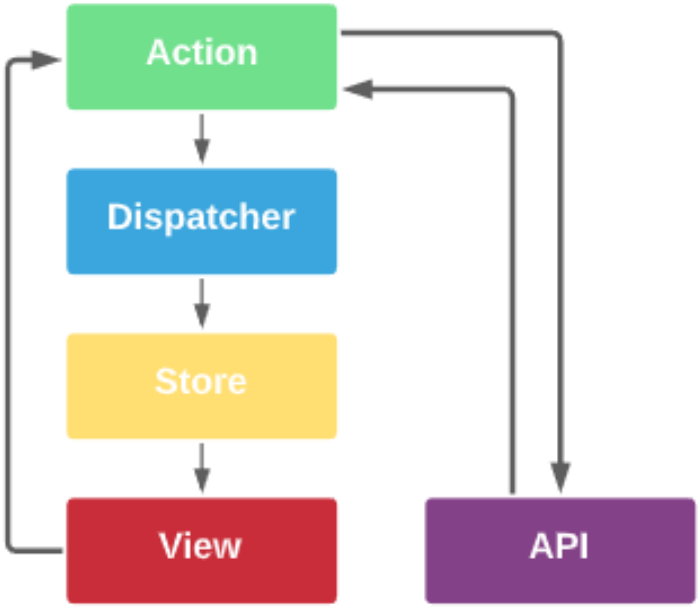

Рисунок 4.8 — Взаємодія компонентів Flux архітектури

Actions для передачі даних сховища через потік Flux — найменш болісний спосіб роботи з даними, що приходять ззовні, або відправляються назовні.

Dispatcher — це менеджер усього цього процесу. Dispatcher розсилає дані Action всім зареєстрованим в ньому обробникам і дозволяє викликати обробники в певному порядку, навіть очікувати оновлень перед тим, як продовжити роботу. Є тільки один Dispatcher, і він діє як центральний вузол усього застосунку. Після цього зміни можуть бути оброблені Сховищем, в результаті чого, стан застосунку буде оновлений, що проілюстровано на рисунку 4.9.

Однією з важливих деталей реалізації Диспетчера є можливість описати залежності і управляти порядком виконання обробників у Сховищах. Отже, якщо для коректного відображення стану один з компонентів програми залежить від іншого, який повинен оновитися перед ним, використовується метод Диспетчера «waitFor». Щоб скористатися цією можливістю, необхідно зберегти значення, що повертається з методу реєстрації в диспетчері, у властивості Сховища «dispatcherIndex».

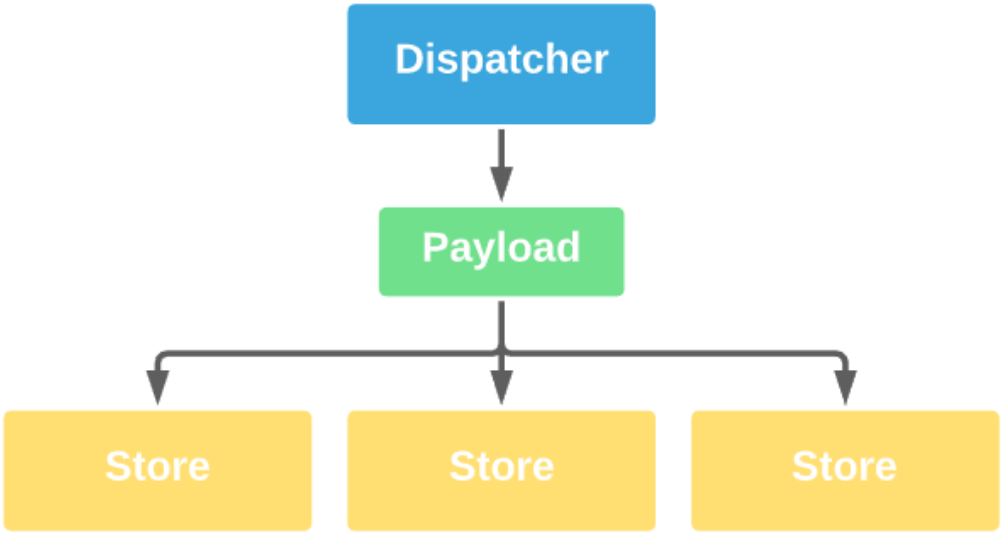

Рисунок 4.9 — Схема роботи «Диспетчера»

Сховища в Flux керують станом певних частин предметної області застосунку. На більш високому рівні це означає, що Сховища зберігають дані, методи отримання цих даних і зареєстровані в диспетчері обробники Дій.

При розробці до сховища були додані можливості EventEmitter з Node.js. Це дозволяє сховищу слухати і розсилати події, що, в свою чергу, дозволяє

компонентам представлення оновлюватися, відштовхуючись від цих подій. Так як представлення слухає подію «change», що створюється Сховищами, воно дізнається про те, що стан застосунку змінився, і час отримати та відобразити актуальний стан.

Також було зареєстровано обробник у Диспетчері за допомогою його методу «register». Це означає, що Сховище слухає оповіщення від Диспетчера. Виходячи з отриманих даних, оператор «switch» вирішує, чи можна обробити Дію. Якщо дію було оброблено, створюється подія «change», і представлення, що підписалися на цю подію, реагують на нього оновленням свого стану, див. рисунок 4.10. Така архітектура дозволяє компонентам залишатися досить акуратними, навіть якщо замість представлень використовувати більш складну логіку.

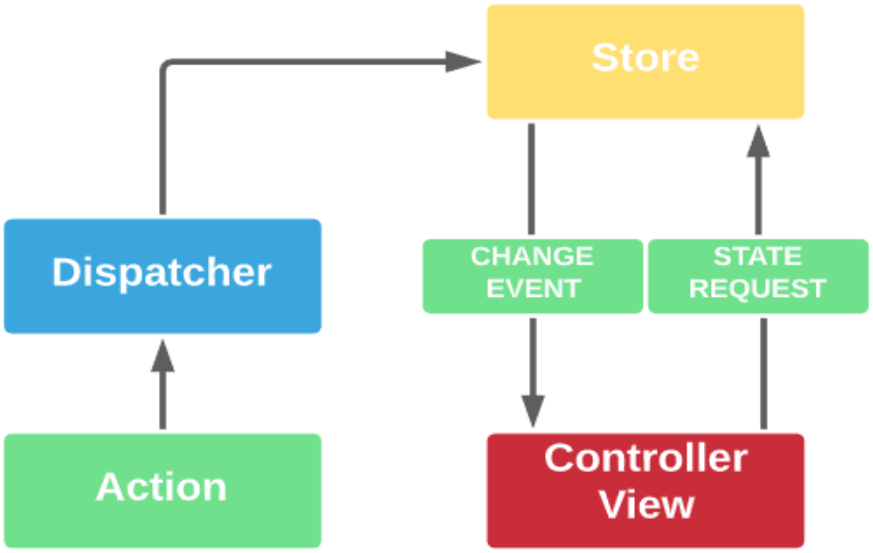

Рисунок 4.10 — Схема оновлення представлення, підписаного на подію «change»

Представлення — це React-компоненти, які підписані на подію «change» і отримують стан застосунку зі Сховищ. Далі вони передають ці дані дочірнім компонентам через властивості, що представлено на рисунку 4.11. Для правильної організації обміну даними між представленнями та поділу даних за сховищами було спроектовано карту шляхів користувача застосунку для клієнтів закладів, див. додаток Ж, адже він має досить розгалуджений систему компонентів.

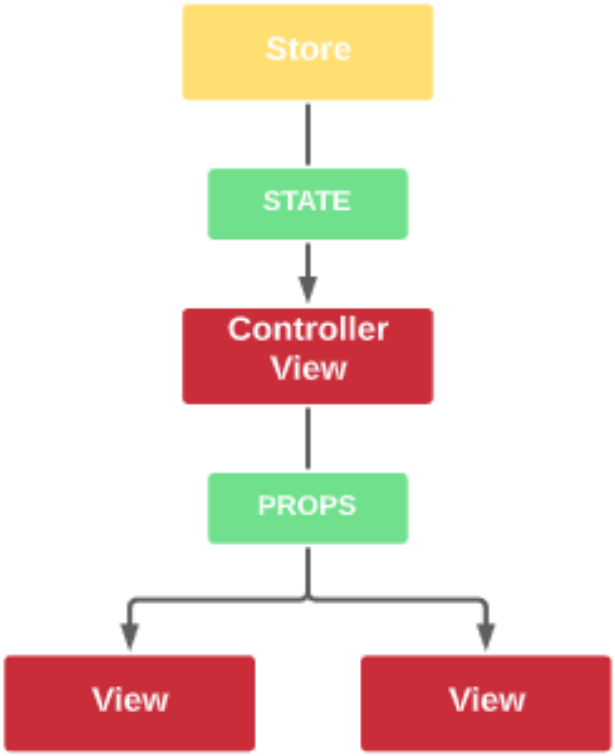

Рисунок 4.11 — Схема передачі властивостей вкладеним компонентам

Використання Flux архітектури породжує у собі використання одного з важливих принципів SOLID архітектури — Dependency Inversion Principle (DIP). DIP — це принцип інверсії залежності, суть якого полягає у тому, що код, який реалізовує високорівневу політику, не повинен залежати від кода, що реалізовує низькорівневі деталі. Навпаки, деталі повинні залежати від політики.

У випадку використання бібліотеки Redux, при реалізації Flux архітектури на верхньому рівні виявляються Reducers і Actions. Вони не мають залежностей і не знають про існування React-компонентів, API і т.п. Вони не залежать навіть від самої бібліотеки Redux. Можна було б навіть написати власну реалізацію бібліотеки Redux, і почати використовувати її замість Redux у застосунку. Діаграма послідовності, наведена у додатку И, відображає застосування Redux на прикладі отримання даних про заклади. У модулях Reducers і Actions навіть не довелося б нічого міняти, оскільки там немає конструкцій на кшталт «import ... from 'redux'».

Redux інкапсулює в собі Стан, даючи лише метод запиту поточного Стану «getState()» і метод «dispatch()» для відправки Дій. Нижче по ієрархії знаходяться селектори. Вони залежать від структури State. Компоненти залежать від структури

State, селектор- і екшен-кріейторів. Крім того, це єдина частина, яка знає про існування DOM-дерева. В результаті отримано поділ застосунку на слої.

4.8.2 Розробка PWA

PWA — це набір технологій розробки веб-застосунків для мобільних пристроїв, що передбачає створення застосунків, які поводять себе, і схожі на нативні. На сьогодні Progressive Web Apps підтримують більшість сучасних браузерів, таких як: Chrome, Safari, Firefox, Edge. Основними технічними складовими при реалізації прогресивного веб-застосунку є Web App manifest, push-сповіщення, Service Worker, App Shell архітектура, HTTPS [13]. На рисунку 4.12 представлено взаємодію компонетів PWA.

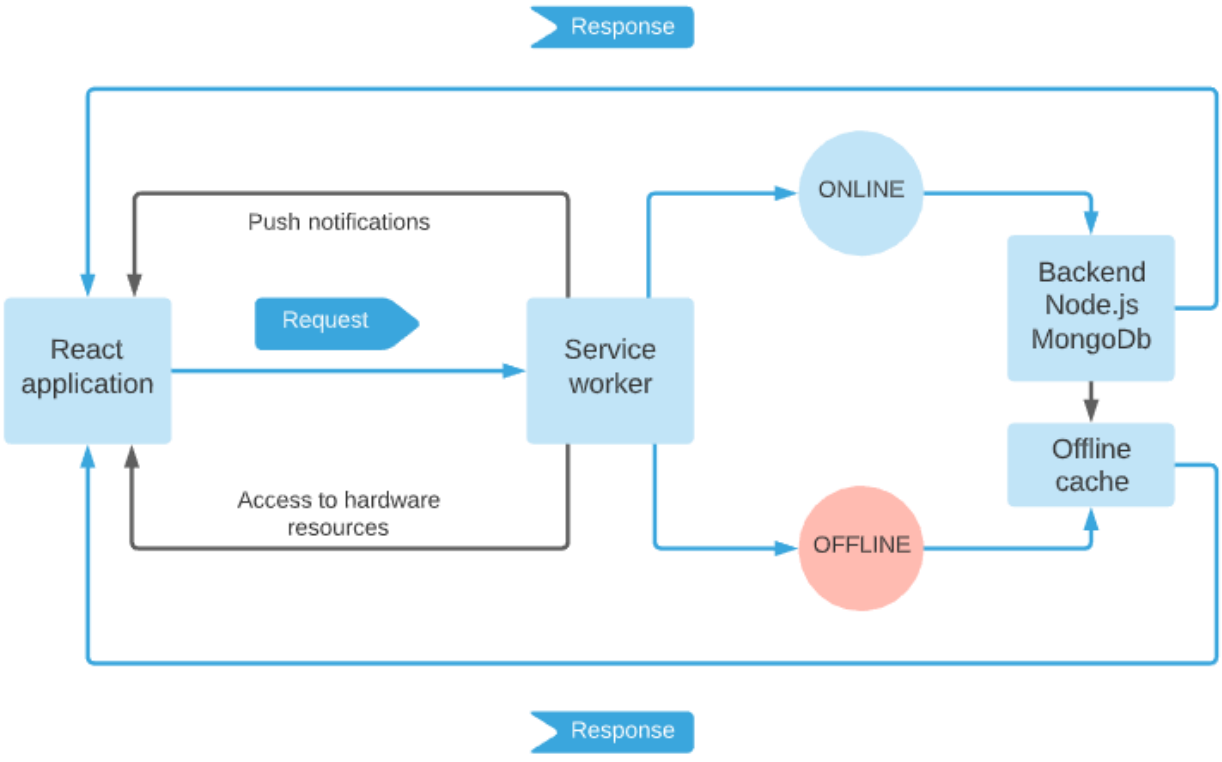

Рисунок 4.12 — взаємодія компонентів PWA

The Web App manifest — це файл JSON, який надає застосунку зовнішній вигляд інтерфейсу нативної програми. За допомогою маніфесту контролюється спосіб

відображення застосунку для користувача (повноекранний режим без видимого рядка URL-адреси) та те, як його запустити. Файл являється централізованим місцем для метаданих веб-програми. Маніфест містить початкову URL-адресу, повне та коротке ім’я застосунку, посилання на піктограми та розміри піктограм, тип та місцезнаходження. Також визначається заставка та колір теми для адресного рядка.

Service worker — ще один технічний елемент, який підтримує одну з основних функцій прогресивних веб-застосунків — можливість роботи в автономному режимі, фонові синхронізації та push-сповіщення, типові для нативних застосунків. Service worker — це файл JavaScript, що працює окремо від програми. Він реагує на взаємодію користувача із застосунком, включаючи мережеві запити, зроблені зі сторінок, які він обслуговує. Оскільки worker працює лише для обробки певної події, термін його служби короткий.

Офлайн-режим роботи. Service worker дозволяє кешувати оболонку програми (інтерфейс), тому вона одразу завантажується при повторних відвідуваннях. Необхідний динамічний вміст оновлюється щоразу, коли з'єднання повертається. Ця механіка дозволяє забезпечити гідну продуктивність програми та покращити взаємодію з користувачем.

Push-сповіщення. Push-сповіщення є ефективним інструментом для повторного залучення користувачів через вміст та оперативні оновлення з веб-сайтів, які їм подобаються. Прогресивні веб-програми можуть надсилати push-сповіщення, навіть коли браузер закритий, а програма неактивна.

Фонова синхронізація. За цю функцію відповідає також Service worker. Він затримує дії, поки не повернеться стабільне підключення. Отже, сервери можуть надсилати періодичні оновлення застосунку, дозволяючи йому оновлюватись після відновлення зв’язку.

Application shell архітектура — передбачає відокремлення статичного вмісту від динамічного. Вона складається з основних елементів інтерфейсу, необхідних для запуску застосунку без підключення.

4.8.3 Організація файлової структури клієнтських стосунків

Одна можливостей React полягає в тому, що ця бібліотека не примушує до суворого дотримання якихось угод, що стосуються структури проекту. Багато що в цьому плані залишається на розсуд розробника. Даний підхід відрізняється від того, який прийнятий, наприклад, у фреймворках Ember.js або Angular. Вони дають розробникам більше стандартних можливостей. У цих фреймворками передбачені і угоди, що стосуються структури проектів, і правила іменування файлів і компонентів.

Обраний підхід до розміщення файлів компонентів по папках, полягає в тому, що ті компоненти, які в контексті програми можна вважати «важливими», «базовими», «основними», розміщуються в окремих папках. Ці папки, в свою чергу, розміщуються в папці components. При цьому «другорядні» компоненти, які використовуються тільки певними «основними» компонентами, розташовуються в тій же папці, що і ці «основні» компоненти. Цей підхід добре показав себе на практиці. Завдяки його застосуванню в проекті з'являється певна структура, але рівень вкладеності папок виявляється не надто великим. Його застосування не призводить до появи чогось на кшталт «../../../» в командах імпорту компонентів, він не ускладнює переміщення по проекту. Цей підхід дозволяє побудувати чітку ієрархію компонентів. Той компонент, ім'я якого збігається з ім'ям папки, вважається «базовим». Інші компоненти, що знаходяться в тій же папці, мають на меті поділу «базового» компонента на частини, що спрощує роботу з кодом цього компонента і його підтримку. Один з мінусів вищеописаного підходу полягає в тому, що його застосування може призвести до появи папок «базових» компонентів, що містять дуже багато файлів. До «базових» папок додаються CSS-файли, файли тестів, безліч підкомпонентів, і інші ресурси — на зразок зображень і SVG-іконок. Все це потрапляє в ту ж папку, що і «базовий» компонент. Але, оскільки файли мають продумані імена, і їх легко можна знайти це можна вважати незначним недоліком.

Файли тестів розміщені в тих же папках, що і файли компонентів. При цьому імена файлів з тестами ідентичні іменам файлів з кодом. До імен тестів, перед розширенням імені файлу, лише додається суфікс «.test».

У такого підходу є кілька сильних сторін:

- він полегшує пошук файлів тестів. Одразу можна зрозуміти те, чи існує тест для компонента, з яким ведеться робота;

- всі необхідні команди імпорту виявляються дуже простими. У тесті, для імпорту тестованого коду, не потрібно створювати структури, що описують, наприклад, вихід з папки __tests__. Подібні команди виглядають досить просто. Наприклад так: «import Auth from './auth'». Якщо є певні дані, які використовуються в ході тесту, вони поміщаються в ту ж папку, де вже лежить компонент і його тест. Коли все, що може знадобитися, лежить в одній папці, це сприяє зростанню продуктивності роботи. Наприклад, якщо використовується розгалужена структура папок і розробник впевнений в тому, що якийсь файл існує, але не може згадати його ім'я, йому доведеться вишукувати цей файл в безлічі вкладених директорій. При пропонованому підході досить поглянути на вміст однієї папки і все стане зрозуміло.

У класичному підході до React (без використання бібліотек для зберігання сховища, як окремої сутності) є два види компонентів — Presentational і Container-Components.

Presentational-Components — презентаційні компоненти. Являють собою, по суті, тільки верстку і відображення даних, які в них передали як властивості.

Container-Components — компоненти, що інкапсулюють логіку роботи компоненту, наприклад, виклики методів життєвого циклу компонента, маніпуляції зі станом компоненту, а також взаємодію зі сховищем. Вони також відповідають за виклик Дії, що відповідає за запит на отримання даних. Повертають мінімум верстки, бо верстка ізольована в «Presentational-Components».

«Create-react-app» генерує базову структуру проекту автоматично, в кореневій директорії містяться файли: «.gitignore», «package.json», «README.md», «yarn.lock». Крім того він створює папки: «public» та «src». рисунок 4.13 представляє у загальному вигляді структуру клієнтських застосунків.

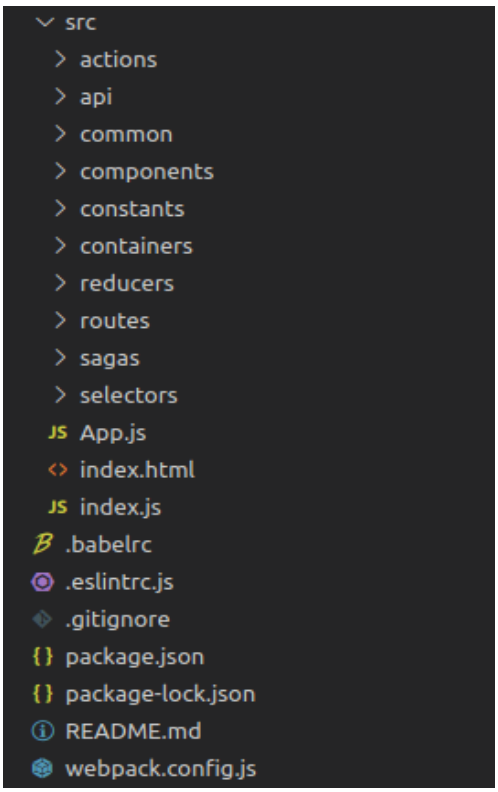

Рисунок 4.13 — загальна структура клієнтського застосунку

Всередині директорії «src», що зберігає у собі вихідний код містяться наступні папки:

- Actions. Тут розміщені файли, що містять функції типу «action-creator», що відповідають за створення об’єкту, що містить назву дії, для передачі у «Redux reducer»;

- Api. Містить код, що відповідає за спілкування з сервером;

- Common. Тут знаходяться допоміжні функції та сервіси, що можуть використовуватись у декількох місцях проекту;

- Components. Тут містяться компоненти проекту;

- Constants. Містить файли з константними значеннями, що використовуються у проекті. Наприклад, назви “екшенів”;

- Containers. Тут розташовані компоненти-обгортки, що по суті являються сторінками застосунку;

- Reducers. Зберігає файли функцій-редьюсерів, що пов’язані з Redux

- Routes. Файли директорії містять код, що відповідає за організацію маршрутизації в проекті;

- Selectors. Містить, так звані, «селектори», що дозволяють простіше організувати отримання даних зі Сховища;

- Sagas. Тут зберігається код «Саг», що є посередником (middleware), надає гнучкі можливості роботи з асинхронними діями у Redux.

4.9 Висновки до розділу 4

Даний розділ присвячений розробці програмного комплексу і містить опис основних архітектурних підходів до реалізації серверної і клієнтської частин. Зокрема описано використання Flux архітектури, що використовується для контролю і обробки змін станів клієнтських застосунків. Також розглянуто основні концепції розробки серверної частини, способи захисту від атак, а також проектування структури бази даних. Наводиться обгрунтування обраних типів зв’язків між документами баз даних а також представлено поля основних документів, що містить база даних. У розділі обгрунтовується організація файлової структури клієнтських застосунків, а також описано основи роботи з технологією PWA.

5 ТЕСТУВАННЯ

Тестування програмного забезпечення — перевірка відповідності між реальною і очікуваною поведінкою програми, що здійснюється на кінцевому наборі тестів, обраному певним чином. У більш широкому сенсі, тестування — це одна з технік контролю якості, що включає в себе активності з планування робіт, проектування тестів, виконання тестування і аналізу отриманих результатів. При тестуванні клієнтських застосунків використовувалися наступні типи тестування:

- функціональне тестування;
- тестування на відмову і відновлення.

5.1 Функціональне тестування

Функціональне тестування — це тестування програмного забезпечення для перевірки реалізованості функціональних вимог, тобто здатність програмного забезпечення за певних умов вирішувати завдання, потрібні користувачам. Функціональне тестування є одним з ключових видів тестування, завдання якого — встановити відповідність розробленого програмного забезпечення вихідним функціональним вимогам. Результати тестування деяких аспектів для обох клієнтських застосунків представлені в таблицях 5.1 та 5.2.

Таблиця 5.1 — Тестування застосунку для пошуку закладів харчування

№	Назва тесту	Вхідні дані	Очікуваний результат	Отриманий результат	Тест пройшов
1	Реєстрація	Незареєстрована в системі електронна пошти	Новий користувач створений в системі, і автоматично	З’явився новий користувач у базі даних, з’явилося повідомлення	Так

№	Назва тесту	Вхідні дані	Очікуваний результат	Отриманий результат	Тест пройшов
			авторизований	про успішну реєстрацію, здійснена автоматична авторизація	
2	Авторизація	Електронна пошта, пароль	Користувач потрапляє до системи під власним аккаунтом	користувач авторизований під своїм акаунтом	Так
3	Збереження іконки застосунку на головному екрані	Відкритий застосунок на мобільному пристрої	Іконку застосунку збережено на головному екрані пристрою	Відповідна іконка застосунку з’явилася на головному екрані	Так
4	Пошук закладів в залежності від місця знаходження	Увімкнене визначення геопозиції пристрою	Відображено результати пошуку, що відповідають поточному місцезнаходженню	Система відобразила усі доступні заклади у заданому радіусі від місця знаходження	Так
5	Застосування	Застосований	Пошук	Система	Так

№	Назва тесту	Вхідні дані	Очікуваний результат	Отриманий результат	Тест пройшов
	фільтрів при пошуці	фільтр заповненості закладів зі значенням макс. 50%	повертає усі доступні заклади із відсотком заповненості =< 50	відобразила усі доступні заклади що відповідають заданому фільтру	
6	Зміна радіусу пошуку	Значення радіусу пошуку відрізняється від встановленого за замовчуванням	Пошук повертає більшу або меншу кількістсть результатів в залежності від змін радіусу пошуку	Кількість відображуваних закладів змінюється в залежності від радіусу пошуку	Так
7	Сортування результатів пошуку	Застосоване сортування результатів	Усі доступні види сортування правильно відпрацювуют ь в обох напрямках	Усі сортування працюють правильно, крім сортування за відстанню до закладу	Частково
8	Відображенн я результатів пошуку на	Готові результати пошуку	Усі знайдені заклади відображають	Усі заклади відображені на карті, але не	Частково

№	Назва тесту	Вхідні дані	Очікуваний результат	Отриманий результат	Тест пройшов
	карті		ся на карті	вміщаються на екрані через неправильне масштабування карти	
9	Перегляд акційних пропозицій	Відкрита сторінка з акційними пропозиціями	Усі доступні акційні пропозицій відображені	Відображені усі доступні акційні пропозиції	Так
10	Відображення акційних пропозицій в результатах пошуку	Відкрита сторінка із результатами пошуку	На картці закладів, що мають спецпропозиції є спеціальна іконка	Заклад, що мають доступні спецпропозиції позначені відповідною іконкою	Так
11	Пошук за ключовими словами	Введена назва закладу у відповідному полі пошуку	Повертаються знайдені результаті, якщо вони знайдені системою	Система відобразила усі знайдені результати, що відповідають пошуку	Так
12	Оформлення замовлення	До корзини замовлення додано	Створюється замовлення	Відобразилось повідомлення про успішне	Так

№	Назва тесту	Вхідні дані	Очікуваний результат	Отриманий результат	Тест пройшов
		страви		замовлення, замовлення з’явилося на сторінці замовлень	

Таблиця 5.2 — тестування застосунку для адміністраторів закладів

№	Назва тесту	Вхідні дані	Очікуваний результат	Отриманий результат	Тест пройшов
1	Зміна інформації про заклад	Заповнена інформація про заклад	Інформація оновилася і збереглася	Повідомлення про успішне збереження інформації, усі зміни відображені	Так
3	Перегляд активних замовлень	Відкрита сторінка замовлень, обрано фільтр активних замовлень	На сторінці відображаються усі доступні активні замовлення	На сторінці відображаються усі замовлення	Частково, фільтр активних замовлень не спрацював
4	Редагувати меню закладу	Відкрита сторінка управління меню	Зміни застосовуються до існуючих пунктів меню, нові пункти	Відображаються повідомлення про здійснені зміни, інформація на	Так

№	Назва тесту	Вхідні дані	Очікуваний результат	Отриманий результат	Тест пройшов
			з’являються при додаванні	сторінці оновлюється відповідно до застосованих дій	
5	Відображення статистики відвідуваності	Відкрита сторінка статистики	Відображаються статистичні згідно обраних фільтрів і способу відображення	При виборі табличного способу відображення або у вигляді діаграми, уся інформація відображається правильно	Так

5.2 Висновки до розділу 5

У розділі було описано тест-кейси для тестування основного функціоналу клієнтських застосунків. Під час тестування було виявлено деякі проблеми в роботі системи, але вцілому система працює справно.

Разом із тестуванням клієнтських застосунків, було протестовано і основний функціонал серверної частини. Усі запити виконувалися у допустимих часових межах, помилок виявлено не було. Також було протестовано виклик захищених роутів, без прав доступу до них. Система повернула відповідну помилку.

6 РОЗРОБКА СТАРТАП-ПРОЕКТУ

6.1 Опис ідеї проекту

При розробці стартап-проекту першим кроком є визначення ідеї самого проекту, а також визначення чим ця ідея може бути корисна потенційним користувачам і які напрямки застосування. У таблиці 6.1 наведено огляд даних пунктів.

Таблиця 6.1 — опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди користувача
	Розробити сервіс, який допоможе знайти заклад харчування, в умовах обмежень, накладених нестабільною епідеміологічною ситуацією.	Значно економиться час пошуку закладу, безпечного і досутпного для відвідування.
	Розробити сервіс, який включатиме структуровану інформаційну базу, інформація в якій адмініструється власниками закладів.	Можна отримати максимально достовірну і актуальну інформацію про заклад перед відвідуванням. Можна адмініструвати і регулярно оновлювати дані про заклад.

Зміст ідеї	Напрямки застосування	Вигоди користувача
система надаватиме можливість здійснення і обробки замовлень їжі.	Платформа для розміщення реклами і спецпропозицій для закладів харчування.	Перегляд усіх доступних спецпропозицій в певному регіоні.
	Платформа онлайн замовлення їжі.	Можна зробити передзамовлення їжі, щоб не очікувати поки страва приготується. Можна зручно зробити замовлення і скориставшись послугою «заберу з собою».

Актуальність проекту полягає в тому, що розроблювана система включатиме функціонал та інформацію, що орієнтовані на пошук закладів, в умовах нестабільної епідеміологічної ситуації, а також надаватиме зручний та універсальний інструмент для здійснення та обробки замовлень їжі. Отже проект є актуальним, через те, що дедалі біль популярною стає послуга «їжа з собою», а також враховуючи накладені обмеження, на сьогоднішній день завдання пошуку закладу для відвідування стає достатньо складним.

Для того щоб мінімізувати ризики, на етапі планування важливо провести порівняльний аналіз продуктів-конкурентів. Важливо створити власну унікальну торгову пропозицію) і відсторонитися від найближчих конкурентів; У таблиці 5.2 наведено порівняння характеристик ідеї проекту із потенційними конкурентами.

Таблиця 6.2. — Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів			W	N	S
		Мій проект	Tripadvisor	Restaurant guru			
1	Трекінг кількості відувачів в реальному часі	Дані про актуальну кількість відвідувачів онвлюються кожні 15 хвилин	Відсутній даний функціонал	Відсутній даний функціонал			+
2	Відображення спецпропозицій та реклами закладів	Реалізовано	Є реклама, але відсутнє відображення спецпропозицій	Не реалізовано			+
3	Пошук закладів з урахуванням карантинних обмежень	Є широкий набів фільтрів для здійснення пошуку	Не реалізовано	Не реалізовано			+

№	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів			W	N	S
		Мій проект	Tripadvisor	Restaurant guru			
4	Підтримка усіх платформ	Реалізовано у вигляді браузерного застосунку із застосуванням технології PWA	Реалізовано у вигляді нативних застосунків для найбільш популярних платформ та у браузері	Реалізовано у вигляді нативних застосунків для найбільш популярних платформ, та як браузерний застосунок		+	
5	Вартість використання	Безкоштовно	Безкоштовно	Безкоштовно		+	
	Вартість підтримки	Відносно невисока, адже необхідно підтримувати лише браузерну версію	Висока, адже необхідно підтримувати браузерну, та мобільні версії під декілька платформ	Висока, адже необхідно підтримувати браузерну, та мобільні версії під декілька платформ			+

6.2 Технологічний аудит ідеї проекту

Важливим аспектом при розробці стартап-проекту є огляд можливих технологій, що можуть бути застосовані при втіленні ідеї у життя і здійсненність окремих ідей проекту. У таблиці 6.3 описані основні ідеї та наведено приклади існуючих технологій для їх реалізації.

Таблиця 6.3 — Технологічна здійсність проекту

№	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Трекінг кількості відувачів в реальному часі	- Інфрачервоний лічильник - Термолічильник - Кінект технології - 2Д Відеолічильник - 3Д Відеолічильник	Усі технології наявні	Усі технології доступні
2	Розробка клієнтського застосунку для пошуку закладів харчування	традиційний веб-застосунок Progressive Web Application Мобільний застосунок	Усі технології наявні	Усі технології доступні
3	Розробка серверної	- Використання мови	Усі технології наявні	Усі технології доступні

№	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
	частини системи	програмування Java - Використання мови програмування C# - Використання мови програмування JavaScript - SDK Firebase		
4	Наявність власної бази даних з інформацією про заклади харчування	- СУБД MongoDB - СУБД Postgres - СУБД MySQL - Firebase realtime database	Усі технології наявні	Усі технології доступні
Обрана технологія реалізації ідеї проекту: для реалізації клієнтського застосунку обрано Progressive Web Application технологію, для реалізації серверної частини обрано мову програмування JavaScript, для збереження даних обрано СУБД MongoDB.				

6.3 Аналіз ринкових можливостей запуску стартап-проекту

Невід'ємною частиною стартап-проекту залежить аналіз можливостей виходу на ринок, адже цей фактор має істотний вплив на успіх проекту. У таблиці 6.4 наведено характеристику ринку.

Таблиця 6.4 — Попередня характеристика потенційного ринку стартап-проекту

№	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	3
2	Загальний обсяг продаж, грн/ум.од	1.5млрд/рік
3	Динаміка ринку (якісна оцінка)	зростає
4	Наявність обмежень для входу (вказати характер обмежень)	немає
5	Специфічні вимоги до стандартизації та сертифікації	немає
6	Середня норма рентабельності в галузі (або по ринку), %	35%

Висновок: на сьогодні ринок насичений потенційними клієнтами, і конкурентами, отже головна задача розробити якісну систему з унікальними функціональними можливостями, яка буде зручною в користуванні і вирішить проблеми з, якими стикаються потенційні клієнти.

Для виявлення можливості отримання доходу від продукту необхідно провести також аналіз можливих клієнтів кінцевого продукту адже у великіій мірі від клієнтів залежить доля стартап-проекту. У таблиці 6.5 описано основні ринкові потреби у сфері, що стосується кінцевого продукту, а також розлядаються освновні характеристики та поведінка цільової аудиторії продукту.

Таблиця 6.5 — Характеристика потенційних клієнтів стартап-проекту

№	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Можливість пошуку закладів харчування в умовах нестабільної епідеміологічної ситуації	Ділові люди, які звикли харчуватися у закладах на регулярній основі. Люди які разово вирішили відвідати заклад. Туристи.	Безкоштовне використання, але із наявністю реклами	- Актуальність отримуваної інформації - Швидкодія - Зручний інтерфейс
2	Можливість трекінгу кількості відвідувачів закладу	Потенційні відвідувачі закладу Власники закладів	Вартість використання продукту та додаткового обладнання, точність підрахунку, вплив на відвідуваність закладу	- Невисока вартість - Легка підтримка - Точність підрахунків - Можливість перегляду статистики
3	Реклама закладів харчування	Власники закладів	Вартість розміщення реклами, охоплення	- Виправдана вартість - Високий рівень охоплення

№	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
			аудиторії, результативність реклами	- Легкий спосіб розміщення рекламних пропозицій
4	Можливість здійснення замовлення їжі	Потенційні відвідувачі закладу Власники закладів	Ціна використання	Зручність, надійність
5	Можливість автоматизованої обробки замовлень їжі	Власники закладів	Ціна використання	Зручність, ефективність, точність
6	Можоивість оформлення передзамовлення їжі у певному закладі на певний час	Відвідувачів закладів	Ціна використання,	Зручність інтерфейсу, надійність сервісу

У будь-якому стартапі наявний високий показник ризику, тому для того щоб передбачити можливі проблеми, що можуть поставити під загрозу успіх проекту важливо розглянути найбільш небезпечеі фактори загроз. У таблиці 6.6 наведено можливі загрози, а також описано варіанти їх подолання.

Таблиця 6.6 — фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
1	Наявність на ринку сильних конкурентів	На ринку присутні декілька сильних конкурентів, продукти яких є досить поширеними серед потенційних користувачів	Розробка унікального функціоналу, який відсутній у конкурентів Розробка максимально зручного у використанні продукту
2	Новий і маловідомий продукт	Низький рівень використання потенційними клієнтами через необізнаність із продуктом	Реклама, піар
3	Важкість виходу на міжнародний ринок	Всі потенційні конкуренти розташовані закордоном. Важкий вихід на контакт із закордонними клієнтами	Почати співпрацю, з сусідніх країн, і далі поступово розвиватися на міжнародні арені.
4	Несприятлива епідеміологічна ситуація	Епідеміологічна ситуація прямо впливає на роботу закладів харчування	Підлаштування під умови ситуації
5	Фінанси	Недостатність коштів для виходу на ринок	Залучення інвестицій

Крім загроз варто звернути увагу на фактори можливостей проекту. У таблиці 6.7 наведено основні з них.

Таблиця 6.7 — Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1	Диференціація витрат	Зменшення витрат за рахунок їх перерозподілу	Зменшення витрат на додаткові, непрофільні задачі.
2	Новизна	Залучення нових клієнтів за рахунок новизни	Розробка функціоналу, що відсутній у конкурентів
3	Гнучкість цін	Зменшення ціни задля збільшення попиту	Введення власних гнучких цін
4	Вільний ринок	Відсутність аналогів, що дозволяють власникам власноруч адмініструвати інформацію, що стосується закладу	Можливість встановлення правил на внутрішньому ринку
5	Прогресування	Постійно розробляти додатковий функціонал, орієнтуючись на потреби клієнтів	Збільшення кількості клієнтів.

Однією з загроз успішності проекту є конкуренція. У таблиці 6.8 розглянуто основні особливості конкурентного середовища, і як ці особливості можуть вплинути на вихід проекту на ринок.

Таблиця 6.8 — Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Вказати тип конкуренції - досконала конкуренція	Існують декілька інших систем, що виконують такі ж функції	Розробка якісного продукту із унікальними функціональними можливостями
2. За рівнем конкурентної боротьби - міжнародний	На ринку пристуні системи, розроблені за кордоном	Розширення кола клієнтів за кордоном, підтримка мультимовності
3. За галузевою ознакою - внутрішньогалузева	Цільова аудиторія продукту стосується однієї галузі	Підвищення якості продукту
4. Конкуренція за видами товарів: - товарно-видова	Продукти схожі за функціональністю	Врахувати недоліки конкурентів, розробити додатковий функціонал
5. За характером конкурентних переваг - нецінова	Різні можливості продукту привертають різну аудиторію	Максимально широкий спектр можливостей
6. За інтенсивністю - марочна	впізнаваний бренд надає високий рівень довіри клієнтів	Приділити увагу розвитку бренду

Для повноти картини важливо проводити всебічний аналіз конкуренції на ринку. У таблиці 6.9 описано результати аналізу галузевої конкуренції.

Таблиця 6.9 — Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	Tripadvisor, Restaurant guru, Google maps, Open Table	Розмір капіталовкладень, ресурси та досвід	Постачальники відсутні	Контроль якості, попит на продукт	Лояльність клієнтів, налагодженість процесів
Висновки:	Tripadvisor та Google maps мають набагато ширший профіль спрямування, тому являються непрямими конкурентами. Restaurant guru — прямий конкурент в Україні. OpenTable — прямий конкурент за кордоном	Є можливості для виходу на ринок адже наявний тільки один прямий конкурент. Строк виходу на ринок становить 1-2 роки. Строки для виходу на міжнародний ринок 4-5 років через наявність конкуренції за кордоном.	У даній галузі постачальники не диктують умов адже продукт не потребує постачальників.	Клієнти частково диктують умови на ринку. Це стосується якості товару та його функціональності	Продукти-замінники створюють певні складнощі, адже користувачі вже звикли до них

За результатами аналізу, що наведені в таблиці, можна зрозуміти, що прямим конкурентом на національному ринку є продукт “Restaurant guru”, оскільки він спеціалізується саме на закладах харчування, а також OpenTable, який популярний за кордоном. Google maps та Advisor є непрямими конкурентами, оскільки мають більш широке спрямування. Щоб вийти на ринок необхідно створити якісний продукт, із наявним у ньому функціоналом, що відсутній у продуктів конкурентів. Вихід на ринок Україне може становити приблизно 1-2 роки, на міжнародний ринок — 4-5 років.

На етапі планування важливо зрозуміти, чи зможе продукти вистояти проти існуючих конкурентів на ринку. Для цього проводиться аналіз його конкурентоспроможності. У таблиці 6.10 наведено обгрунтування основних факторів конкурентоспроможності.

Таблиця 6.10 — Обгрунтування факторів конкуроентоспроможності

№	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Багатофункціональність	Конкуренти не мають можливості трекінгу кількості відвідувачів, та відображення відсотку заповненості закладу
2	Орієнтованість на епід.ситуацію	Продукти конкурентів не мають орієнтації на умови нестабільної епідеміологічної ситуації
3	Актуальністть даних	Актуальність даних у продукті забезпечується тим, що вони заповнюються власниками закладів. Така можливість відсутня у конкурентів.
4	Інтерфейс	Привабливий та зручний інтерфейс

Для більш точного визначення конкурентоспроможності продукту, необхідно виконання порівння цього переваг та недоліків відносно інснуючих конкурентів. У таблиці 6.11 наведено рейтингову оцінку факторів конкурентоспроможності продукту з позиції огляду продуктів існуючих конкурентів.

Таблиця 6.11 — Порівняльний аналіз сильних та слабких сторін системи

№	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з агрегатором						
			–3	–2	–1	0	+1	+2	+3
1	Багатофункціональність	15					+		
2	Орієнтованість на епідеміологічну ситуацію	20	+						
3	Можливість замовлення їжі з собою	20	+						
4	Інтерфейс	18				+			

У таблиці 6.12 наведено результати SWOT-аналізу проекту, що є невід’ємною частиною етапу проектування.

Таблиця 6.12 — SWOT-аналіз стартап-проекту

Сильні сторони: орієнтовність на епідеміологічну ситуацію, Багатофункціональність	Слабкі сторони: Відсутність клієнтської бази, низька репутація на початку, відсутність досліджень поведінки клієнтів
Можливості: попит, вихід на міжнародний ринок, прогресування	Загрози: відсутність достатньої кількості фінансових ресурсів, низький рівень інтересу до продукту

При виході продукту на ринок, не завжди вдається правильно визначити стратегію його впровадження, тому завжди необхідно планувати альтернативні варінти впровадження. У таблиці 6.13 описані альтернативні підходи до ринкового впровадження продукту.

Таблиця 6.13 — Альтернативи ринкового впровадження стартап-проекту

№	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Розробка мобільних застосунків для найпопулярніших платформ	70%	6 місяців
2	Розширення функціоналу продукту	65%	3 місяці

6.4 Розробка ринкової стратегії проекту.

Основоположним кроком, від якого залежить успіх усього проекту є розробка ринкової стратегії. Ринкова стратегія — це набір правил, якими керується менеджмент організації при прийнятті управлінських рішень. Це загальний комплексний план, призначений для забезпечення здійснення місії і досягнення господарських цілей організації.

Ринкова стратегія передбачає ефективне досягнення цілей економічними методами і засобами. Для розробки правильної ринкової стратегії, важливо оцінити потенційні групи споживачів на предмет готовності сприймати продукт, а також орієнтовний попит серед них. У таблиці 6.14 описано важливі аспекти характеристик основної споживацької аудиторії.

Таблиця 6.14 — Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Великі мережі закладів харчування				
2	Заміські ресторани				
3	Немережеві заклади харчування				
Які цільові групи обрано: 1, 2, 3					

В якості цільових груп потенційних споживачів було обрано усі доступні групи, адже усі вони готові сприйняти продукт, і залучення кожної з груп буде сприяти поширенню продукту серед усіх можливих сегментів споживачів.

Після впровадження продукту на ринок, важливо продовжувати його розвивати і постійно покращувати взаємодію з ним, тому необхідно визначити стратегію подальшого розвитку, ще на етапі планування. У таблиці 6.15 описуються ключові моменти обраної стратегії розвитку.

Таблиця 6.15 Визначення базової стратегії розвитку

№	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку*
1	Розробка мобільних застосунків для найпопулярніших платформ	Стратегія спеціалізації	Покращення зручності взаємодії, збільшення функціональних можливостей, висока універсальність	Стратегія диференціації

Для успішного функціонування на ринку, важливо ще до виходу на ринок визначитись із поведінкою на ньому. У таблиці 6.16 розглянуто базові аспекти поведінки на ринку по відношенню до конкурентів.

Таблиця 6.16 — Визначення базової стратегії конкурентної поведінки

№	Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки*
1	Ні	Так	Ні	Стратегія виклику лідера

Важливим пунктом успішного впровадження продукту на ринок, є правильно обрана стратегія позиціонування продукту. Для формування такої стратегії, першим кроком є визначення вимог користувачів до продукту, і на основі них — визначення конкурентних позицій. У таблиці 6.17 наведено основні моменти, що пов’язані з цим.

Таблиця 4.17 — Визначення стратегії позиціонування

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту
	Якість, точність, простота у використанні, швидкодія, доступність	Стратегія лідерства по витратах	Функціональніст, гнучкість, актуальність інформації	По іміджу, позиціювання за сферою застосування

6.5 Розроблення маркетингової програми стартап-проекту

Одну з найбільш важливих ролей у впровадженні продукту на ринок посідають реклама та маркетинг. Адже без проведення успішної маркетингової кампанії, навіть сильний та конкурентоспроможний продукт може зазнати невдачі. Для складення правильного маркетингового плану необхідно спершу визначити усі вигоди, що надає продукт і на основі цих вигод, виділити переваги у порівнянні з конкурентами. У таблиці 6.18 визначаються основні потреби потенційних користувачів на основі яких формуються переваги концепції продукту.

Таблиця 6.18. Визначення ключових переваг концепції потенційного товару

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами
1	Можливість пошуку закладів харчування в умовах нестабільної епідеміологічної ситуації	Наявність у системі фільтрів, що стосуються пошуку закладів в залежності від їх показників безпечності, в умовах нестабільно епідеміологічної ситуації	Конкуренти не мають подібного функціоналу
2	Моніторинг кількості відвідувачів закладу	Дозволяє відслідковувати та відображати інформацію стосовно кількості відвідувачів закладу в реальному часі, що може позитивно сказатися на відвідуваності закладу	Конкуренти не мають подібних можливостей
3	Реклама закладів харчування	Дозволяє власникам закладів розміщувати рекламу та спецпропозиції та акції	Деякі конкуренти мають інтеграцію реклами, але вони не дають можливості для розміщення спец. пропозицій та акцій
4	Здійснення і обробка замовлення їжі	Дозволяє користувачам замовити їжу «на винос» або зробити передзамовлення, щоб довго не чекати приготування	Конкуренти не мають даної можливості

Важливим етапом при розробці маркетингової стратегії є огляд трирівневої моделі продукту. Даний огляд наведено у таблиці 6.19.

Таблиця 6.19 — Опис трьох рівнів моделі товару

<table>
<tr><td>Рівні товару</td><td colspan="3">Сутність та складові</td></tr>
<tr><td>І. Товар за задумом</td><td colspan="3">Система для пошуку закладів харчування</td></tr>
<tr><td rowspan="5"></td><td>Властивості/ характеристики</td><td>М/Нм</td><td>Вр/Тх /Тл/Е/Ор</td></tr>
<tr><td>Властивості: Система складається із односторінкового застосунку для власників закладів і PWA застосунку для потенційних відвідувачів закладів.</td><td>НМ</td><td>ТХ/ТЛ</td></tr>
<tr><td colspan="2">Якість: швидкість та коректність у використанні, логічність та простота інтерфейсу користувача.</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td colspan="2">Марка:</td></tr>
<tr><td rowspan="2"></td><td colspan="3">До продажу: базова версія</td></tr>
<tr><td colspan="3">Після продажу: постійна модернізація</td></tr>
<tr><td colspan="4">За рахунок чого потенційний товар буде захищено від копіювання: створення торгової марки та впровадження авторського права.</td></tr>
</table>

При розробці стартап проекту важливо одразу визначитись із ціновою політикою, для подальшого прорахунку ліквідності. У таблиці 6.20 розглянуто аспекти формування цінових меж для продукту.

Таблиця 6.20 — Визначення меж встановлення цін

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
	—	—	Не можливо визначити однозначно	5%-10% від замовлення, 100-300$/місяць для закладу

Для будь-якого продукту важливо мати можливості для успішного збуту/ поширення серед користувачів. У таблиці 6.21 описано специфку збуту для розроблюваного стартапу.

Таблиця 6.21 — Формування системи збуту

№	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
	Орієнтація на нових клієнтів	- розміщення застосунку на власному маркеті; - реклама застосунку на головних сторінках;	1	Залучена система збуту через сторонніх посередників

При проведенні маркетингової кампанії важливим аспектом є комунікація з потенційними клієнтами. У таблиці 6.22 розглядаються основні моменти формування маркетингових комунікацій.

Таблиця 6.22 — Концепція маркетингових комунікацій

№	Специфіка поведінки цільових клієнтів	Канали комунікацій клієнтів	Ключові позиції для позиціонування	Завдання рекламного повідомлення	Концепція звернення
	клієнти є власниками закладів харчування клієнти користуютьс я послугами закладів харчування	Соціальні мережі, Інтернет, YouTube, телефонний зв’язок	Легкість використання системи 2. Надання права повного адмініструванн 3.Можливість удосконалень	Звернути увагу на зручність та багатофункціон альність системи, економію часу.	SMM

6.6 Висновки до розділу 6

У розділі проведено огляд основних ідей проекту. Був здійснений аналіз ринку, що включав огляд тенденцій характерних для даної галузі. Також було визначено основних конкурентів, сильні та слабкі сторони продукту у порівнянні з ними. До сильних сторін можна віднести: орієнтованість на умови епідеміологічної ситуації, ширші функціональні можливості у порівнянні з конкурентами, зручний інтерфейс. В результаті дослідження можна зробити висновки, що проект має можливості для виходу на ринок, але для цього необхідно зробити якісний продукт, що виділятиметься унікальним функціоналом серед конкурентів.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було розроблено програмний комплекс, що дозволяє здійснювати пошук закладів харчування в умовах нестабільної епідеміологічної ситуації, а також представляє платформу для передзамовлення їжі у закладах і впровадження послуги «їжа з собою». Для розробки серверної частини, а також клієнтських частин, було використано мову JavaScript, що виправдало себе значною мірою, адже використання однієї мови для обох сторін значно спростило та прискорило процес розробки. Обрана СУБД MongoDB у зв’язці з ODM Mongoose повністю задовольнили вимоги проекту. SPA інтерфейси розроблені з використанням бібліотеки React працюють швидко, та справляються позитивний досвід користування. Отже, можна зробити висновки, що загалом всі технології, а також архітектиурні підходи, які були використані при розробці системи задовольнили поставленим вимогам.

Крім технічної частини, було проведено огляд систем-аналогів, основні переваги і недоліки яких були враховані при розробці власного рішення. Також було проведено тестування системи, в основному методом чорного ящику, що дозволило виявити ряд недоліків у роботі системи, та виправити їх.

Також було розроблено стартап-проект, у якому було розглянуто основні бізнес-можливості реалізації продукту на сьогоднішньому ринку, а також розглянуто фактори конкурентоспроможності та інноваційності, що містить в собі система.

Подальший розвиток системи буде направлений на доопрацювання нереалізованих частин та аспектів, що були визначені у вимогах до застосунку, а також розробку більш широкого спектру функцій, серед яких: можливість онлайн-оплати, можливість бронювання столиків та інше.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Stellman A. Applied Software Project Management / A. Stellman, J. Greene. – Sebastopol: O'Reilly Media, 2018. – 303 с.
2. Freeman A. Pro React 16 / Adam Freeman. – London: Apress, 2019. – 750 с.
3. Lauret A. The Design of WEB APIs / Arnaud Lauret. – Shelter Island: Manning publications, 2019. – 389 с.
4. Ashwini A. Should You Use NoSQL Or SQL Db Or Both? [Електронний ресурс] / Amit Ashwini // Medium. – 2017. – Режим доступу до ресурсу: https://medium.com/swlh/should-you-use-nosql-or-sql-db-or-both349cb26c9add#:~:text=You%20can%20store%20huge%20amounts,advance%20with%20document%2Dbased%20databases.
5. Coron T. What is Sass? Your guide to this top CSS preprocessor [Електронний ресурс] / Tammy Coron // Creativebloq. – 2020. – Режим доступу до ресурсу: https://www.creativebloq.com/web-design/what-is-sass-111517618#:~:text=Sass%20(which%20stands%20for%20'Syntactically,to%20create%20style%20sheets%20faster..
6. Rendle R. BEM 101 [Електронний ресурс] / Robin Rendle // csstricks. – 2016. – Режим доступу до ресурсу: https://css-tricks.com/bem-101/.
7. Anh P. Virtual DOM: its definition and benefits that you must know [Електронний ресурс] / Phuong Anh // ArrowHitech. – 2018. – Режим доступу до ресурсу: https://www.arrowhitech.com/virtual-dom-its-definition-and-benefits-that-you-must-know/.
8. Vanali M. Understand basic Webpack from scratch [Електронний ресурс] / Marco Vanali // Medium. – 2019. – Режим доступу до ресурсу: https://medium.com/swlh/understand-basic-webpack-from-scratch-6a1976565ae0.
9. Web API design [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design.
10. Полторак В.П. Криптографічний захист даних в цифрових інформаційних системах (Частина 3) / Полторак В.П. // Телеком. Військовий зв’язок.

Спеціальний випуск №2, 2019. - К.: Softpress. hi-Tech.ua, Жовтень, 2019. - с. 64-67. Мова публікації: українська.

11. Полторак В.П. Криптографічний захист даних в цифрових інформаційних системах (Частина 2) / Полторак В.П. // Телеком. Військовий зв'язок. Спеціальний випуск №1, 2019. - К.: Softpress. hi-Tech.ua, Квітень, 2019. - с. 81-85. Мова публікації: українська.
12. Полторак В.П. Криптографічний захист даних в цифрових інформаційних системах (Частина 1) / Полторак В.П. // Телеком. Військовий зв'язок. Спеціальний випуск №2, 2018. - К.: Softpress. hi-Tech.ua, Жовтень, 2018. - с. 98-104. Мова публікації: українська.
13. 23 рекомендации по защите Node.js приложений [Електронний ресурс] // Medium. – 2018. – Режим доступу до ресурсу: https://medium.com/webbdev/23-%D1%80%D0%B5%D0%BA%D0%BE%D0%BC%D0%B5%D0%BD%D0%B4%D0%B0%D1%86%D0%B8%D0%B8-%D0%BF%D0%BE-%D0%B7%D0%B0%D1%89%D0%B8%D1%82%D0%B5-node-js-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9-e3fbc348f92.
14. Deutsch S. 5 React Architecture Best Practices [Електронний ресурс] / Sebastian Deutsch. – 2018. – Режим доступу до ресурсу: https://www.sitepoint.com/react-architecture-best-practices/.
15. Pilsch A. Events in Flux: Software Architecture, Detractio, and the Rhetorical Infrastructure of Facebook [Електронний ресурс] / Andrew Pilsch // Computers and Composition. – 2020. – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/abs/pii/S8755461520300451?via%3Dihub.
16. Progressive Web Apps: Core Features, Architecture, Pros and Cons [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://www.altexsoft.com/blog/engineering/progressive-web-apps/.